

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank) X	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED final	
4. TITLE AND SUBTITLE Optimal Real-Time Control of Stochastic, Multipurpose, Multireservoir Systems		5. FUNDING NUMBERS X	
6. AUTHORS x C. Russ Philbrick			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) x Stanford University		AFRL-SR-BL-TR-98- 0032	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) x AFOSR/NI 110 Duncan Avenue, Room B115 Bolling Air Force Base, DC 20332-8080		10. SPONSORING/MONITORING AGENCY REPORT NUMBER X	
11. SUPPLEMENTARY NOTES X			
12a. DISTRIBUTION AVAILABILITY STATEMENT x Approved for Public Release		12b. DISTRIBUTION CODE X	
13. ABSTRACT (Maximum 200 words) <p>This thesis presents new systems-analysis methods that are appropriate for complex, non-linear systems that are driven by uncertain inputs. These methods extend the ability of discrete dynamic programming (DDP) to system models that include six or more state variables and a similar number of stochastic variables. This is accomplished by interpolation and quadrature methods that have high-order accuracy and that provide significant computational savings over traditional DDP interpolation and quadrature methods.</p> <p>These new methods significantly improve our ability to apply DDP to large-scale systems. Using these methods, DDP can solve a variety of systems analysis problems without resorting to the simplifying assumptions required by other stochastic optimization methods. This is demonstrated in the application of DDP to problems with as many as seven state variables. Of particular interest, this thesis applied DDP to the practical problem of conjunctively managing groundwater and surface water. Moreover, the applications also demonstrate that DDP can be a powerful planning tool, such as when evaluating a range of capacity expansion alternatives.</p>			
14. SUBJECT TERMS X		15. NUMBER OF PAGES X	
		16. PRICE CODE X	

19980116 051

OPTIMAL REAL-TIME CONTROL OF
STOCHASTIC, MULTIPURPOSE, MULTIRESERVOIR SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF CIVIL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
CIVIL ENGINEERING

C. Russ Philbrick
December 1996

© Copyright by Charles Russell Philbrick, Jr. 1996
All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy

Peter K. Kitanidis (Principal Advisor)
Professor, Department of Civil Engineering

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy

George B. Dantzig
Professor Emeritus, Department of Operations Research
and Engineering Economics Systems

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy

Steven M. Gorelick
Professor, Department of Geological and
Environmental Sciences

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy

Gerd Infanger
Senior Research Engineer, Department of Operations
Research and Engineering Economics Systems

Approved for the University Committee on Graduate Studies

ABSTRACT

This thesis presents new systems-analysis methods that are appropriate for complex, non-linear systems that are driven by uncertain inputs. These methods extend the ability of discrete dynamic programming (DDP) to system models that include six or more state variables and a similar number of stochastic variables. This is accomplished by interpolation and quadrature methods that have high-order accuracy and that provide significant computational savings over traditional DDP interpolation and quadrature methods.

These new methods significantly improve our ability to apply DDP to large-scale systems. Using these methods, DDP can solve a wide variety of systems analysis problems without resorting to the simplifying assumptions required by other stochastic optimization methods. This is demonstrated in the application of DDP to problems with as many as seven state variables. Of particular interest, this thesis applies DDP to the practical problem of conjunctively managing groundwater and surface water. Moreover, the applications also demonstrate that DDP can be a powerful planning tool, such as when evaluating a range of capacity expansion alternatives.

ACKNOWLEDGMENTS

Support for this work has been provided by the Laboratory Graduate Fellowship Program of the Department of Energy and by the United States Air Force Laboratory Graduate Fellowship Program of the Air Force Office of Scientific Research (AFOSR). Computer resources used in this work have been provided by National Science Foundation Grant BSC-8957186 and the Hewlett Packard Foundation.

TABLE OF CONTENTS

Acknowledgments.....	vii
Table of Contents	ix
List of Tables	xix
List of Illustrations	xxiii
 Chapter 1. Introduction	 1
A. Motivation	2
1. Evolving Challenges in Water Resources Planning and Management	2
B. An Example: Conjunctive Management of Groundwater and Surface Water	3
1. Benefits of Conjunctive-Use	4
2. The Management Problem of Conjunctive-Use	6
3. A Solution	6
C. Scope of the Dissertation	7
1. Background	8
2. New Methods	8
3. Applications	9
 Chapter 2. Analysis of Reservoir Systems	 11
A. Using Simulation and Optimization in Reservoir System Management	12
B. The Impact of Stochastic Inputs	13
1. An Example of Management Without Uncertain Inputs	13
2. The Effect of Uncertain Inputs on Reservoir Control	14
3. The Purpose of Feedback Control	16
4. The Impact of Reducible and Irreducible Uncertainty	17
C. Developing Systems Analysis Framework	18
1. Mathematical Model of a System	19
Decision Variables	19
State Variables	19
Stochastic Variables	20
Dynamics	21
Constraints	21

Feasibility of Control Decisions with Stochastic Constraints.....	22
Arguments for Using Limited Foresight	23
(1) Feasibility of Control Decisions	24
(2) Limited Computer Resources	24
(3) Convergence of Solutions with Short Stages	24
(4) Unboundedness	24
2. Value Model of a System	24
Accuracy of Value Models	25
Value Function Description	26
Performance in the Presence of Uncertain Inputs	26
3. Optimization of System Performance	27
Difficulty in Identifying Performance for All Stages	28
Description of System Performance	30
Chapter 3. Review of Optimization Methods For Stochastic Dynamic Control	31
A. Current Status of Systems Analysis Applied to Water-Supply Management	32
Summary of Optimization Methods	33
B. Forecast-Based Methods	36
Deterministic Feedback Control (DFC)	36
First-Order Analysis Methods	37
Chance Constraints	38
Linear-Quadratic Control	39
Advantage of Forecast-Based Methods	39
Simplifying Assumptions of Forecast-Based Methods	40
Application of Forecast-Based Methods to Water-Supply Management	41
C. Parametric Methods	42
Regression	43
Neural Networks	44
Advantage of Parametric Methods	44
Simplifying Assumptions of Parametric Methods	45
Application of Parametric methods to Water-Supply Management	45
D. Stochastic Dynamic Programming Methods	47
Parametric Dynamic Programming	48
Discrete Dynamic Programming	49

Stochastic Dual Dynamic Programming	50
Static Dynamic Programming	51
Advantage of SDP	51
Simplifying Assumptions of SDP	52
Application of SDP to Water-Supply Management.....	53
 Chapter 4. Dynamic Programming	 55
A. What is Dynamic Programming?	56
B. Discrete Dynamic Programming	59
1. Illustration of the Last Stage Subproblem.....	59
2. Illustration of the Remaining Subproblems Using Recursion	61
C. Limitations of Discrete Dynamic Programming	61
1. Exponential Growth with State Dimension	62
2. Techniques to Reduce Exponential Growth.....	64
3. Cost-To-Go Interpolation Methods.....	66
D. Multilinear Interpolation	68
1. Linear Interpolation in 1-D	68
2. Linear Interpolation in Multiple Dimensions.....	69
3. Local Coordinate System	70
 Chapter 5. New Hermite Interpolation Methods	 73
A. Characteristics of an Efficient Interpolation	73
B. Advantages of Hermite Interpolation	74
C. Characteristics of Weighting Functions	76
1. Requirements to Preserve Node Values and Gradients in One Dimension	76
2. Requirements to Preserve Node Values and Gradients in Multiple Dimensions	77
3. Additional Requirements to Produce Continuous and Smooth Interpolating Functionals in Multiple Dimensions	78
D. Original Hermite Interpolation Method	79
1. The Weighting Functions	80
2. Analysis of the Original Hermite Interpolation Method	81
E. New First-Order Hermite Interpolation Method	82
1. Hermite Interpolation in One Dimension.....	83
2. Hermite Interpolation in Multiple Dimensions	84

F. Convexity of the First-Order Method	90
1. Convexity of One-Dimensional Interpolation.....	90
2. Convexity of Multi-Dimensional Interpolation	91
G. A Hermite Interpolation Method with Continuous Second Derivatives	92
1. The Weighting Functions.....	93
2. The One-Dimension Approximating Functional	94
3. The Two-Dimension Approximating Functional.....	95
4. Accuracy and Convexity of Hermite Interpolation with Continuous Second Derivatives	96
H. A Second-Order Hermite Interpolation Method	98
1. Shorthand Notation	99
2. Weighting Function for the Second Derivatives.....	100
I. Computational Efficiency of Methods.....	101
Chapter 6. Analysis of Gradient Dynamic Programming.....	105
A. The Series of Multi-Reservoir Test Problems.....	106
1. The Four-Reservoir Test Problem.....	106
2. Formulation of the Multi-Reservoir Test Problems	108
B. Computational Effort to Solve the Series of Multi-Reservoir Test Problems.....	109
1. Standardization of Computational Time	110
Linear Interpolation.....	111
First-Order Hermite Interpolation	112
Second-Order Hermite Interpolation	113
2. Growth in Interpolation Effort with State Dimension	114
3. Growth in Number of Interpolations for Each Discrete State.....	116
Searches Per Node for Each Interpolation Method.....	117
Interpolations Per Search for Each Interpolation Method	118
4. Growth in Total Effort with State Discretization.....	118
C. Accuracy of the Series of Multi-Reservoir Test Problems.....	122
1. Error Reduction with State Discretization	122
2. Error Analysis	125
3. Solutions of the Four-Reservoir Test Problem	126
D. Net Performance of the Different Interpolation Methods	127
1. A Reformulation of the Results	129
E. Concluding Remarks	131

Chapter 7.	Methods of Numerical Integration to Evaluate Expected Values	133
A.	Effect of Stochastic Variable Dimension on Computation	133
1.	Effort to Evaluate Expected Values	134
2.	Evaluation of Expected Values by Numerical Integration.....	135
3.	Classical Numerical Integration: The Trapezoidal Rule	136
4.	Examples of Past Efforts to Evaluate Expected Values.....	136
5.	Discretization of Stochastic Variables in Past Efforts	138
B.	The Application of Gaussian Quadrature to Discrete Dynamic	
Programming.....		139
1.	Mathematical Form of Gaussian Quadrature	140
2.	Compatibility of Gaussian Quadrature and Hermite Interpolation	140
3.	Sources of Error in Applying Gaussian Quadrature to Discrete	
Dynamic Programming		141
Piecewise Nature of Cost-To-Go Approximations		142
Changing Control Decisions		142
Non-Polynomial Cost and Transition Functions.....		143
Constraints		143
C.	Identification of Gaussian Quadrature Weights and Abscissas.....	144
1.	Gaussian Quadrature with Normal Distributions	144
2.	Gaussian Quadrature with Three-Parameter Gamma Distributions ..	146
3.	Gaussian Quadrature with Lognormal Distributions	147
4.	Gaussian Quadrature with Arbitrary Distributions	147
Chapter 8.	Analysis of Gaussian Quadrature.....	151
A.	Gaussian Quadrature Accuracy in Estimating the Expected Cost-To-Go	151
1.	The Stochastic Models	151
2.	Error Versus Stochastic Discretization	152
3.	Error Bias of Gaussian Quadrature	154
4.	Comparison with a Heuristic Quadrature Method	156
B.	Impact of Piecewise Nature of Cost-To-Go Approximations on	
Solution Accuracy		157
1.	Quadrature Error Versus Interpolation Error	158
2.	Quadrature Error with State Discretization.....	158
Chapter 9.	Caution in Real-Time Operation of Reservoir Systems.....	163
A.	Motivation	163

B. Background	164
C. System Models	165
1. Four-Reservoir Model with Uncorrelated Streamflows (Model A)...	166
2. Four-Reservoir Model with Correlated Streamflows (Model B).....	167
3. Four-Reservoir Model with Higher-Order Cost Function (Model C)	169
D. Results	169
1. Results for the Uncorrelated Flow Model (A)	170
2. Results for the Correlated Flow Model (B)	174
3. Results for the Non-Quadratic Penalty Model (C).....	178
E. Concluding Remarks on the Cautious Management of Reservoir Systems ..	182
Chapter 10. Valuation of Water Resources	185
A. Background on Market Prices	185
1. An Ideal Market	186
2. Non-Ideal Water Markets.....	188
B. Assumptions	190
1. Constant Elasticity	192
2. Availability of Water Supplies.....	193
3. Effect of Timing on Rationing Costs	194
4. Externalities	194
C. Rationing Cost Function.....	195
D. Application to an Example System	197
1. The Example Water supply System	197
2. Water Prices	198
3. Demand Elasticity	199
4. Assessment of Rationing Costs.....	199
5. Application to the Example System.....	200
Chapter 11 Optimal Conjunctive-Use Operations and Plans	203
A. Introduction	203
B. Problem Description	205
1. The EBMUD System	205
2. Proposed Aquifer Storage	206
3. System Model	206
4. Demand, Streamflow, and Storage	208
5. Value Model.....	209

Shortage Cost	209
Pumping Cost	210
Recharge Cost	211
C. Evaluation of Operations and Plans	211
D. Results for Real-Time Operations	213
1. Supply Policy	214
2. Allocation Between Surface and Subsurface Storage	215
Pumping Policy	215
Recharge Policy	216
Allocation	217
3. Downstream-Release Policy	217
4. Expected Cost of System Operations	218
5. Cost-To-Go	219
E. Results for Capacity Expansion	221
1. Benefits of Groundwater Development	222
2. Benefits of Conjunctive Development of Groundwater and Surface Water	224
3. Impact of Initial Conditions on Results	225
4. Impact of Discount Rate on Results	227
F. Concluding Remarks on the Conjunctive Management of Surface and Groundwater Storage	230
Chapter 12. Conclusions	233
Appendix A. Summary of Notation	237
1. Notation	237
2. Equations	239
1. System Model	240
2. Value Model	240
3. General Solution	240
4. Dynamic Programming Solution	240
5. Interpolation	240
Appendix B. Computer Code for Discrete Dynamic Programming and Enhancements	241
1. Simplified Flow Chart	241
2. Include Files for Common Storage of Data	242

Include File I.SIZEALLO	242
Include File I.SIZEPROB	243
Include File I.XNODES	243
Include File I.SPECW	244
Include File I.FNODES	244
Include File I.CONTROL	244
Include File I.SPECNOW	245
Include File I.CUBE	245
Include File I.PERFORM	245
3. Interpolation Subroutines	245
Subroutine INT_FUNC	246
Subroutines INT_HC2, INT_HC1, and INT_LIN	247
Subroutine CUBE_ID	252
Subroutine CUBEVAL2	255
4. Optimization Subroutines	258
Subroutine OPT_SOLV	258
Subroutine OPT_FUNC	263
Subroutine OPT_NPSL	264
Subroutine OPT_POLY	270
Subroutine OBJFUN	272
Subroutine OBJVAL	274
Subroutine OBJ_CALC	274
Subroutine COST_PEN	278
5. Main Subroutine and Accessories	280
Subroutine DYNPROG	280
Subroutine MODELALL	289
Subroutine MODELSTG	292
Subroutine NODE_VAL	302
Subroutine ADJ_MOD	311
Subroutine IDNOW	328
Subroutine SIZETEST	329
6. Model Specification Subroutines	330
Subroutine CALLDP	330
Subroutine SPECPROB	330
Subroutine SPECU	332
Subroutine SPECX	333

Subroutine SPECW	334
Subroutine TRANSX	337
Subroutine SPECLCON	339
Subroutine COST_NOW	340
Subroutine FINALCTG	342
Subroutine SPECF	343
Subroutine OUTSTAGE	343
Subroutine OUTFINAL	344
References	347

LIST OF TABLES

Table 1B1. Advantages and Disadvantages of Subsurface and Surface Reservoirs.....	5
Table 2C1. Common Measures of Performance, Control, and State	19
Table 3A1. Optimization Methods Used for Stochastic Dynamic Control	35
Table 5C1. Weighting-Function Requirements for Interpolation in One-Dimension ...	77
Table 5C2. Weighting-Function Requirements for Interpolation in Multiple Dimensions.....	78
Table 5C3. Weighting-Function Requirements for Continuity and Smoothness on a Regular Grid	79
Table 5C4. Weighting-Function Requirements for Continuity and Smoothness on an Irregular Grid.....	79
Table 5G1: Weighting-Function Requirements for Second-Derivative Continuity on an Irregular Grid.....	93
Table 5H1. Second-Derivative Weighting-Function Requirements for Continuity and Smoothness on an Irregular Grid	98
Table 5I1. Distribution of Effort for One Interpolation of the Cost-To-Go (in Flops)..	103
Table 5I2: Total Flops For Each Evaluation of the Cost-to-Go (per Code)	103
Table 6A1. Definition of Multi-Reservoir Problems	109
Table 6B1. Standard Computational Times per Stage of Linear Interpolation	112
Table 6B2. Standard Computational Times per Stage of First-Order Hermite Interpolation	113
Table 6B3. Standard Computational Times per Stage of Second-Order Hermite Interpolation	114
Table 6B4. Comparison Between Actual and Hypothetical Growth in Interpolation Effort	116
Table 6B5. Breakdown of Effort for Each Discrete State of a Subproblem	117
Table 6B6. Impact of State Discretization on Standard Computational Time of Second-Order Hermite Interpolation	119
Table 6B7. Impact of State Discretization on Standard Computational Time of First-Order Hermite Interpolation	120
Table 6B8. Impact of State Discretization on Standard Computational Time of Second-Order Hermite Interpolation	121
Table 6C1. Impact of State Discretization on Accuracy of Linear Interpolation	124

Table 6C2. Impact of State Discretization on Accuracy of First-Order Hermite Interpolation	124
Table 6C3. Impact of State Discretization on Accuracy of Second-Order Hermite Interpolation	124
Table 6C4. Error Reduction Obtained From Halving the Discretization Interval of Linear Interpolation.....	125
Table 6C5. Impact of State Discretization on Accuracy of First-Order Hermite Interpolation	126
Table 6C6. Impact of State Discretization on Accuracy of Second-Order Hermite Interpolation	126
Table 6C7. Solution Convergence with Finer Discretization for the Four-Reservoir Problem When $\mathbf{x}_{t_1} = [6,6,6,6]^T$	127
Table 6C8. Solution Convergence with Finer Discretization for the Four-Reservoir Problem When $\mathbf{x}_{t_1} = [1,1,1,1]^T$	127
Table 6D1. Standardized Time per Stage to Achieve 10% and 1% Average Absolute Relative Error	130
Table 6D2. Ratio of Standardized Time Using Linear and Hermite Interpolation to Achieve 10% and 1% Average Absolute Relative Error	130
Table 7C1. Gaussian Quadrature Abscissa Locations and Weights for Standard Normal Distribution	146
Table 7C2. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(2.0, 0.5)$	149
Table 7C3. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(4.0, .75)$	149
Table 7C4. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(2.0, 1.0)$	150
Table 7C5. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(4.0, 1.5)$	150
Table 8A1. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization (Normally Distributed Inflows)	153
Table 8A2. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization (Lognormally Distributed Inflows)	153
Table 8A3. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization (High-Variance Lognormally Distributed Inflows).....	154
Table 8A4. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization (Normally Distributed Inflows)	155

Table 8A5. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization (Lognormally Distributed Inflows)	155
Table 8A6. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization (High-variance Lognormally Distributed Inflows).....	155
Table 8A7. Abscissa Location and Weights for Heuristic Quadrature Method	157
Table 8A8. Error (% AARE) of a Heuristic Quadrature Method and Gaussian Quadrature.....	157
Table 8B1. Total Error (% AARE) of Gaussian Quadrature with State Discretization (Normally Distributed Inflows and First-Order Hermite Interpolation).....	158
Table 8B2. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Normally Distributed Inflows and First-Order Hermite Interpolation).....	159
Table 8B3. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Normally Distributed Inflows and Second-Order Hermite Interpolation).....	160
Table 8B4. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Lognormally Distributed Inflows and First-Order Hermite Interpolation).....	160
Table 8B5. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Lognormally Distributed Inflows and Second-Order Hermite Interpolation).....	160
Table 8B6. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (High-Variance Lognormally Distributed Inflows and First-Order Hermite Interpolation).....	161
Table 8B7. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (High-Variance Lognormally Distributed Inflows and Second-Order Hermite Interpolation)	161
Table 9C1. Parameters of Stochastic Variables for Uncorrelated Flow Model.	166
Table 9C2. Model of Correlated Stream Flows.	168
Table 9C3. Parameters of Stochastic Variables for Correlated Flow Model.	168
Table 9D1. Mean, Minimum, and Maximum of Cost Distributions.....	169
Table 11B1. Variables of the Simple Conjunctive-Use Model	208
Table 11E1. Variables for Capacity Expansion of the Simple Conjunctive-Use Model	222
Table AA1. Notation for System Model	237

Table AA2. Conventionally Defined and Non-Specific Parameters, Variables, and Functions	238
Table AA3. Notation for Value Model and Optimal Solution.....	238
Table AA4. Notation for Effort of Discrete Dynamic Programming	238
Table AA5. Notation for Interpolation of the Cost-To-Go Function.....	239
Table AA6. Notation for Numerical Integration of the Expected Cost-To-Go	239
Table AA7. Notation for Rationing-Cost Function	239

LIST OF ILLUSTRATIONS

Figure 2B1. Example Trajectories for Regulation of Reservoir Level	14
Figure 2B2. Effect of Unexpected Storm on Regulation	15
Figure 2B3. Effect of Real-Time Control and Better Foresight on Regulation	16
Figure 2B4. Effect of Caution on Regulation	16
Figure 3B1. Discrete States and State Trajectories for the Last-Stage Subproblem.....	60
Figure 3B2. Sample Streamflow Distribution.....	60
Figure 3B3. Discrete Estimate of the Last-Stage Cost-To-Go Function	60
Figure 3B4. Discrete States and Sample State Trajectories for the Second-to-Last Subproblem	61
Figure 4C1. Discrete States for a 1-D Problem	63
Figure 4C2. Discrete States for a Multidimensional Problem	63
Figure 4C3. Cost Function Accuracy for Various State-Variable Discretizations	66
Figure 4C4. Comparison of Methods to Improve Interpolation Accuracy	68
Figure 4D1. Hypercubes of One, Two, and Three Dimensions.....	70
Figure 4D2. Local Coordinate System for Gridded 2-D Domain.....	72
Figure 5E1. 1-D Linear and First-Order Hermite Interpolation of the Function $F(x) = x^{-1}$	84
Figure 5E2. 2-D Weighting Function $\phi(\eta)$ Applied at a Node	87
Figure 5E3. 2-D Derivatives of Weighting Function $\phi(\eta)$ Applied at a Node	87
Figure 5E4. 2-D Weighting Function $\psi(\eta)$ Applied at a Node	88
Figure 5E5. 2-D Derivatives of Weighting Function $\psi(\eta)$ Applied at a Node	88
Figure 5E6. 2-D First-Order Hermite Interpolation of the Function $F(\mathbf{x}) = (x_1x_2)^{-1}$	89
Figure 5G1. 1-D Hermite Interpolation with Continuous Second Derivatives of the Function $F(x) = x^{-1}$	95
Figure 5G2. 2-D Hermite Interpolation with Continuous Second Derivatives of the Function $F(\mathbf{x}) = (x_1x_2)^{-1}$	96
Figure 6A1. Illustration of the Four-Reservoir Control Problem	107
Figure 6A2. Multi-Reservoir Systems	108
Figure 6D1. Trade-Off Between Accuracy and Effort per Stage of Linear Interpolation	128
Figure 6D2. Trade-Off Between Accuracy and Effort per Stage of First-Order Hermite Interpolation	128

Figure 6D3. Trade-Off Between Accuracy and Effort per Stage of Second-Order Hermite Interpolation	129
Figure 6D4. Growth in Effort with Number of State Variables to Achieve 10% and 1% Average Absolute Relative Error	130
Figure 7A1. Probability Distributions for Various Discretizations of a Stochastic Variable	135
Figure 9C1. Uncorrelated-Flow Scenario for Models A and C (Example 1).	167
Figure 9C2. Uncorrelated-Flow Scenario for Models A and C (Example 2).	167
Figure 9C3. Correlated-Flow Scenario for Model 2 (Example 1).	168
Figure 9C4. Correlated-Flow Scenario for Model 2 (Example 2).	168
Figure 9D1a. Distribution of Costs for DFC Policy (Model A).	171
Figure 9D1b. Distribution of Costs for DDP Policy (Model A).	171
Figure 9D1c. Distribution of Cost Differences for DFC Versus DDP (Model A).	171
Figure 9D2a. Release and Storage for 1st Reservoir (Model A, Example 1).	172
Figure 9D2b. Release and Storage for 2nd Reservoir (Model A, Example 1).	172
Figure 9D2c. Release and Storage for 3rd Reservoir (Model A, Example 1).	172
Figure 9D2d. Release and Storage for 4th Reservoir (Model A, Example 1).	172
Figure 9D3a. Release and Storage for 1st Reservoir (Model A, Example 2).	173
Figure 9D3b. Release and Storage for 2nd Reservoir (Model A, Example 2).	173
Figure 9D3c. Release and Storage for 3rd Reservoir (Model A, Example 2).	173
Figure 9D3d. Release and Storage for 4th Reservoir (Model A, Example 2).	173
Figure 9D4a. Distribution of Costs for DFC Policy (Model B).....	175
Figure 9D4b. Distribution of Costs for DDP Policy (Model B).	175
Figure 9D4c. Distribution of Cost Differences for DFC Versus DDP (Model B).....	175
Figure 9D5a. Release and Storage for 1st Reservoir (Model B, Example 1).	176
Figure 9D5b. Release and Storage for 2nd Reservoir (Model B, Example 1).....	176
Figure 9D5c. Release and Storage for 3rd Reservoir (Model B, Example 1).....	176
Figure 9D5d. Release and Storage for 4th Reservoir (Model B, Example 1).....	176
Figure 9D6a. Release and Storage for 1st Reservoir (Model B, Example 2).	177
Figure 9D6b. Release and Storage for 2nd Reservoir (Model B, Example 2).....	177
Figure 9D6c. Release and Storage for 3rd Reservoir (Model B, Example 2).....	177
Figure 9D6d. Release and Storage for 4th Reservoir (Model B, Example 2).....	177
Figure 9D7a. Distribution of Costs for DFC Policy (Model C).....	179
Figure 9D7b. Distribution of Costs for DDP Policy (Model C).	179
Figure 9D7c. Distribution of Cost Differences for DFC Versus DDP (Model C).....	179
Figure 9D8a. Release and Storage for 1st Reservoir (Model C, Example 1).	180

Figure 9D8b. Release and Storage for 2nd Reservoir (Model C, Example 1).....	180
Figure 9D8c. Release and Storage for 3rd Reservoir (Model C, Example 1).....	180
Figure 9D8d. Release and Storage for 4th Reservoir (Model C, Example 1).....	180
Figure 9D9a. Release and Storage for 1st Reservoir (Model C, Example 2).	181
Figure 9D9b. Release and Storage for 2nd Reservoir (Model C, Example 2).....	181
Figure 9D9c. Release and Storage for 3rd Reservoir (Model C, Example 2).....	181
Figure 9D9d. Release and Storage for 4th Reservoir (Model C, Example 2).....	181
Figure 10A1. Example Demand and Supply Functions.....	187
Figure 10A2. Consumer and Producer Surplus	187
Figure 10A3. Impact of Non-Market Price on Consumer and Producer Surplus	188
Figure 10A4. Producer Surplus and Consumer Surplus for Water Supply	190
Figure 10B1. Elastic ($\alpha = -2$), Isoelastic ($\alpha = -1$), and Inelastic ($\alpha = -0.5$) Demand Functions	191
Figure 10B2. Impact of Storage on Market Equilibrium Scenarios	193
Figure 10B3. Impact of an Alternate Supply on Market Equilibrium Scenarios.....	193
Figure 10C1. Graphical Estimation of the Total Benefit of Water Consumption	196
Figure 10C2. Rationing Cost Versus Fraction of Normal Supply for $\alpha = -0.5$	197
Figure 10D1. Estimate of Rationing Cost for a Typical Family	201
Figure 11B1. The Simple Conjunctive-Use System	207
Figure 11D1. Supply Policy: Release (TAF per Year) to Users as a Function of Available Water	214
Figure 11D2. Pumping and Recharge Policies: Transfers (TAF per Year) from and to Groundwater as a Function of Available Water.....	215
Figure 11D3. Release Policy: Release (TAF per Year) Downstream as a Function of Available Water	218
Figure 11D4. Expected Total Cost (Million \$) as a Function of Available Water Using Foresight of Current Year's Inflows	219
Figure 11D5. "Cost-To-Go": Expected Total Cost (Million \$) from Future Inflows as a Function of Initial Storage Levels	220
Figure 11E1. Expected Annual Cost (Million \$) for Different Levels of Groundwater Development with Initially Full Reservoirs.....	223
Figure 11E2. Expected Annual Cost (Million \$) for Different Levels of Conjunctive Development with Initially Full Reservoirs	225
Figure 11E3. Expected Annual Cost (Million \$) for Different Levels of Groundwater Development with Initially Empty Reservoirs	226

Figure 11E4. Expected Annual Cost (Million \$) for Different Levels of
Conjunctive Development with Initially Empty Reservoirs 227

Figure 11E5. Pumping and Recharge (TAF per year) with a Zero Discount Rate 229

Figure 11E6. Expected Annual Cost (Million \$) for Different Levels of
Conjunctive Development with Zero Discount Rate 229

Figure AB1. Simplified Flow Chart for the DDP code..... 242

CHAPTER 1.

INTRODUCTION

The challenges of water resources management are increasing, complicating the efforts of managers who rely on traditional heuristic methods of system operation and planning. Not only are objectives of water management changing, but the means by which managers can achieve these objectives are dramatically different from what they were twenty years ago. Conflict is common as increasing demands must be balanced by consideration for environmental qualities without the ability to tap new resources. Conservation, conjunctive use, desalination, reclamation and other techniques for meeting demands have replaced dam construction. With increasing competition for limited water resources, the solution of water resources management problems will continue to become more complex.

Mathematical modeling and optimization, also known as "systems analysis," are increasingly useful methods in coping with these challenges. With advances in algorithms and computing power, we have greatly expanded our ability to develop practical solutions for realistic water management problems. In contrast, the limited ability of earlier methods often required that practitioners resort to simplistic solutions developed for drastically simplified water management problems. As a result, application of these solutions was often found to be unsatisfactory [*Rogers and Fiering, 1986*].

This thesis is an effort to further expand the abilities and acceptance of systems analysis in the management of water systems. It is my belief that these methods are becoming increasingly relevant and practical because of rapid changes in water management. In particular, this thesis demonstrates the application of systems analysis to managing systems that combine groundwater and surface water, commonly known as "conjunctive use." To accomplish this, this thesis presents analysis methods that allow solution of stochastic optimization problems of greater complexity than previously possible, not just for water resources management, but also for many other management problems that fit the general mathematical form.

A. MOTIVATION

This effort began with some straight-forward systems operation questions that turned out not to have straight-forward answers. A local water supply agency, the East Bay Municipal Utility District (EBMUD) of Oakland California, has been considering structural and operational options to reduce the agency's susceptibility to water shortages in its growing district. While working on the groundwater storage option of this project, I became interested in understanding the operational effects of adding groundwater storage to the existing surface reservoir system. In particular, I was interested in comparing benefits of groundwater storage with benefits of additional surface reservoir capacity. The first question that needed answering was how operation of the groundwater component should differ from existing surface reservoirs. A direct comparison of capacities was inappropriate because a groundwater component would likely face significant constraints on rate of recharge and extraction while also offering significantly larger storage volumes. Not until this first question was answered could the second question, how to compare the benefits of adding groundwater storage or additional surface storage, be answered correctly.

It was a desire to answer these questions that lead to the use of optimization and, in particular, the application of dynamic programming (DP) methods. However, limitations on available DP methods frustrated these efforts, prompting work to overcome these limitations. This thesis is a summary of successful methods developed to overcome these limitations and of their application to answer the initial questions. It does not include the many failed methods and dead-end paths that seem to be inevitable in such a journey.

Though the specific problem of conjunctive-use management led me to this work, the broader potential of this work has provided the motivation to address the problem to the degree presented in this thesis. By developing general methods to answer the above questions, I hope to have developed methods applicable to many other water resource management problems that we face.

1. Evolving Challenges in Water Resources Planning and Management

Increasingly rapid changes in supply, demand, and system configuration are requiring managers to reevaluate the management of their systems. Moreover, the objectives of water management are changing and system managers must now consider ill-defined "costs" and "benefits" of qualities such as impacts on fisheries, riparian ecosystems, scenery, recreation, and water quality. To meet these changing demands,

systems increasingly employ management methods that include water conservation, system redundancy, prioritizing deliveries, and integration of alternate sources such as groundwater, desalinated water, and recycled water. Managers find it increasingly difficult to develop operating rules as these rapid changes reduce the base of experience available to guide operations.

Many water-management practices result from years of experience operating water systems. Drawing upon this experience, managers have developed operating rules that generally do an acceptable job of reducing the risk and expected cost arising from water shortages and floods, while increasing the benefits of, say, hydropower generation [Bredehoeft *et al.*, 1995; Kelman *et al.*, 1990].

However, with changes in supply, demand, and system configuration, operating rules must be updated if system performance is to be maintained. With gradual changes, managers may be able to update these rules incrementally without much degradation in performance. Rapid changes, however, may require that managers update operating rules much more dramatically with potentially large degradation in performance.

Even when changes are gradual, the failure to periodically update operating rules can result in dramatic changes when their weaknesses are highlighted in a crisis. For example, the severe 1976-77 drought in western North America resulted in drastic revision of operating rules for many water supply agencies. Many had failed to recognize their susceptibility to drought as demands grew in their service areas. As a result, many agencies suffered severe shortages; and, in response, many of these have become much more cautious in their allocation of water supplies. This has been demonstrated by preemptive rationing initiated during the more recent periods of drought and by some expensive new projects that have included reclaiming waste water, desalination, and water conservation.

B. AN EXAMPLE: CONJUNCTIVE MANAGEMENT OF GROUNDWATER AND SURFACE WATER

Systems that conjunctively manage groundwater and surface water, often called "conjunctive use" systems, present examples where managers may find difficulty in developing operating rules. Dam construction has become difficult—if not impossible—because the best available sites have been used and environmental considerations have eliminated many remaining potential sites [Lettenmaier and Burges, 1979]. Also, public perception has turned against further dam construction. Consequently, aquifer storage has become more attractive.

However, conjunctive use is still a relatively novel management method, and managers may hesitate to take advantage of conjunctive-use benefits. Although both ground and surface water resources are widely used for water supply, these sources are most often managed independently [Lettenmaier and Burges, 1979]. In part, this may be because management of conjunctive-use systems can be difficult. Efficient management may not be possible using common-sense or heuristic methods because of the different capabilities and limitations of storage in surface reservoirs and aquifers. Without appropriate management policies, managers may be discouraged by increased uncertainty in the risks and costs of conjunctive-use development.

1. Benefits of Conjunctive-Use

In contrast to dam construction, the public seems more willing to accept the use of aquifers for storing water. Though this is partially due to the less obvious impact that aquifer storage has on modifying the environment, there are real advantages to aquifer storage, especially in areas where pumping has already depressed groundwater levels. Aquifer storage can help restore groundwater levels and thus reduce the costs of salt-water intrusion, land subsidence, and pumping lifts. Also, aquifer storage may avoid evaporation and seepage losses associated with surface reservoirs, and avoid engineering risks and costs associated with dam construction. Aquifer storage is not a panacea, however; it also has its own associated costs, such as from pumping and conflicts over land use and water rights.

From an operational perspective, conjunctive use is a possible method for improving water supply reliability and efficiency. Willis and Yeh [1987, p. 241] recognize that, "By controlling the total water resources of a region, conjunctive use planning can increase the efficiency, reliability, and cost-effectiveness of water use, particularly in river basins with spatial or temporal imbalances in water demands and natural supplies."

Many of the largest water supply systems traditionally have relied entirely on surface reservoir storage [van der Leeden *et al.*, 1990, pp. 319-325]; thus, there may be many opportunities for their improvement through utilization of aquifer storage. It is likely that initial efforts to employ aquifer storage will be more cost effective than expansion of surface-reservoir storage. Lettenmaier and Burges [1979] found that, under certain assumptions, developing aquifer storage as a buffer against variations in stream flow was about an order of magnitude cheaper than developing surface storage.

Systems that integrate aquifer storage and surface reservoirs should be designed to enhance the advantages and mitigate the disadvantages of surface and subsurface storage

(Table 1A1). Management policies can use these differences to increase the reliability and reduce the operating cost of conjunctive-use systems. As *Burges and Maknoon* [1975, p. 1] point out, "Whenever multiple sources of water with different characteristics, as is the case with groundwater and surface water systems, are available, it may be possible to develop an operating strategy which exploits the different characteristics of the sources." For example, conjunctive-use systems should be better at simultaneously meeting water supply and flood control objectives by combining the long-term storage capability of groundwater with the short-term surge capacity of surface reservoirs.

Table 1B1. Advantages and Disadvantages of Subsurface and Surface Reservoirs

Subsurface Reservoirs	Surface Reservoirs
<u>Advantages</u>	<u>Disadvantages</u>
1. Many large capacity sites available	1. Few new sites available
2. Slight to no evaporation loss	2. High evaporation loss even in humid climate
3. Require little land area	3. Require large land area
4. Slight to no danger of catastrophic structural failure	4. Danger of catastrophic failure
5. Uniform water temperature	5. Fluctuating water temperature
6. High biological purity	6. Easily contaminated
7. Serve as conveyance systems and avoids need to establish right-of-way	7. Water must be conveyed by canal or pipeline
<u>Disadvantages</u>	<u>Advantages</u>
1. Water must be pumped	1. Water may be available by gravity flow
2. Storage and conveyance use only	2. Multiple use
3. Water may be mineralized	3. Water generally of relatively low mineral content
4. Minor flood control value	4. Maximum flood control value
5. Limited flow at any point	5. Large flows
6. Power head usually not available	6. Power head available
7. Difficult and costly to investigate, evaluate, and manage	7. Relatively easy to evaluate, investigate, and manage
8. Recharge opportunity usually dependent on surplus surface flows	8. Recharge dependent on annual precipitation
9. Recharge water may require expensive treatment	9. No treatment required of recharge water
10. Continuous expensive maintenance of recharge areas or wells	10. Little maintenance required of facilities

Source: U. S. Bureau of Reclamation, *Ground Water Manual*, U. S. Department of the Interior, 1977 (referenced by *van der Leeden et al.*, 1990, p. 648).

2. The Management Problem of Conjunctive-Use

Because few conjunctive-use systems exist, we have little experience managing them. We cannot easily develop heuristic operating rules that are efficient, and this increases the expected expense and risk of developing new systems. Under these conditions, planners and managers are hesitant to develop conjunctive-use systems because of uncertain performance and unfamiliarity with these systems.

Many managers' first exposure to conjunctive-use operations likely will occur when aquifer storage is added to existing surface reservoir systems. In contemplating the addition of groundwater to their system, managers currently appear to hold two extreme views. On one hand, they view aquifer storage as equivalent to a surface reservoir. On the other hand, they view aquifer storage as a back-up: to be recharged only when other reservoirs are full, and to be depleted only when other reservoirs are empty.

Both views fail to recognize the different capabilities and limitations of surface and subsurface storage (Table 1B1). In particular, surface reservoirs and subsurface reservoirs have different storage capacities, recharge and depletion rate limits, and operating costs. If we base management rules on the notion that groundwater provides only a backup supply or that subsurface storage is equivalent to surface storage, conjunctive-use systems will perform little better than sole-source systems that rely only on groundwater or only on surface water. On the other hand, if we develop management rules that take advantage of the capabilities and avoid the limitations of each storage mechanism, we may significantly improve system efficiency and reliability.

3. A Solution

Systems analysis can help managers understand the use and benefits of adding aquifer storage to a reservoir system. Systems analysis involves mathematical modeling, simulation, and optimization to determine operating rules that maximize system performance. Through such an analytic approach, we can resolve conflicting views about the use of aquifer storage. Also, by determining operating rules in advance of actual operations, we can better anticipate the benefits and costs of conjunctive-use systems and can better compare alternative system designs. Greater ability to anticipate benefits and costs can reduce the uncertainty associated with the development of conjunctive-use systems.

A systems analysis approach can also allow us to incorporate a variety of criteria and information into water system management. These include a wider consideration of system costs such as externalities that affect third parties or the environment. These may

also include constraints that incorporate legal or public policy requirements. Systems analysis can incorporate these other criteria much more efficiently than heuristic methods and can allow easier evaluation of changes. In addition, systems analysis can allow water management agencies to determine a monetary value for proposed changes in system design or operation, aiding in public policy consideration of conflicting objectives. Consequently, agencies and policy makers can evaluate management alternatives and assess their tradeoffs with greater objectivity.

C. SCOPE OF THE DISSERTATION

This dissertation develops systems analysis methods and applies them to a few management problems, concluding with the problem of conjunctive-use. The methods and applications are both presented in generic terms to facilitate application to other water resource management problems. Indeed, the methods are valid for many optimal control problems in other fields, particularly those concerned with resource management. These include energy distribution, financial planning, chemical engineering, or any field that poses problems that fit the general mathematical form of stochastic DP. Application of these methods is most useful when considering processes driven by uncontrolled and uncertain inputs, just as reservoir management is driven by the uncontrolled and uncertain inputs of precipitation and stream flow.

We can see the difficulty in controlling such systems by previewing the challenge presented by reservoir management. The control of stochastic dynamic systems is complicated by our need to determine controls for these systems in advance of essential knowledge. In particular, the regulation of stream flows for water supply requires that we make allocation decisions before we know future streamflows with certainty. As a result, we can only hope to make allocation decisions that are the best on average after considering all future stream flows that are possible. Also, though we may determine a current "best" allocation, we want to update allocation decisions as future stream flows become known; therefore, we cannot identify in advance a best trajectory that identifies the system's future condition.

The chapters of this thesis are divided into three groups. The first group lays the groundwork for presentation of developments and applications discussed in the remaining chapters. The second group develops and analyses new systems analysis methods. The final group applies these methods to a conjunctive-use problem and considers other practical considerations associated with application of systems analysis methods.

1. Background

Chapter Two presents background on systems analysis and stochastic optimization methods. This chapter describes the impact that uncertain inputs have on systems analysis and provides notation that will be used in the rest of this thesis.

Chapter Three reviews optimization methods that can be used when system models contain uncertain inputs. This chapter briefly describes each method and its limitations, and describes a few water-resource applications.

Chapter Four presents the optimization method of discrete dynamic programming (DDP) as an appropriate method for analysis of reservoir systems. This chapter discusses the limitations of DDP and provides additional notation and equations used in this thesis. The development of multilinear DDP is presented as an illustration of DDP and as an introduction to gradient dynamic programming (GDP) presented the following chapter.

2. New Methods

Chapter Five develops new interpolation methods for application in a GDP algorithm. These interpolation methods use both values and gradients for more accurate approximation of a cost-to-go function used by DDP. Development is guided by the need to produce a highly efficient numerical code, and a standard measure of computational effort is used to anticipate the actual performance of each method.

Chapter Six applies GDP to a range of test problems. This chapter compares GDP with multilinear DDP by contrasting the computational time and accuracy of the two techniques. The analysis validates the expected performance discussed in the previous chapter and identifies the potential for using Gradient DP in problems with as many as six to eight state variables.

Chapter Seven introduces efficient numerical-integration (i.e., quadrature) methods to calculate expected values in stochastic DDP problems. These methods use Gaussian quadrature for numerical integration of the stochastic expected-value function. Development is guided again by the need to produce efficient numerical code. Gaussian quadrature is an independent method that can improve efficiency almost as dramatically as GDP. Though independent, efficient quadrature is an important parallel development to efficient interpolation. In many practical applications, models require a number of stochastic variables that increases in parallel with the number of state variables.

Chapter Eight applies Gaussian quadrature with GDP and multilinear DDP using the test problems of Chapter Six. This chapter illustrates the high-order accuracy of Gaussian quadrature and contrasts Gaussian quadrature with other methods that have

been applied in water resources problems. The analysis validates the expected performance discussed in the previous chapter and the potential for solving problems with numerous stochastic variables.

3. Applications

Chapter Nine applies the GDP code to test the value of caution in reservoir management. GDP is applied to a series of reservoir control problems characterized by different levels of caution required for optimal control. From these applications, we can observe that management based most-likely forecasts of future reservoir inflows is insufficiently cautious and can result in poor system performance, particularly under extreme conditions. Chapter Nine also demonstrates the value of a systems analysis approach that uses DDP because of its ability to establish management policies that are appropriately cautious.

Chapter Ten considers our ability to quantify the value of water for urban users. Optimal control is meaningless without an explicit statement of the values used to evaluate system performance. This can be difficult when applying systems analysis to practical problems. This chapter presents a reasonable method that can be used to identify water-supply values.

Chapter Eleven applies GDP to a simple conjunctive-use problem. From this application, we can observe some of the management practices that contribute to the efficient control of a conjunctive-use system and some of the benefits expected from adding groundwater storage to existing surface reservoir systems. In a broader sense, this application also illustrates the practical benefit of using DDP to identify real-time control policies. Moreover, this application demonstrates our ability to use DDP to assess a wide range of planning options in an integrated approach. This integrated approach allows us to simultaneously identify optimal control policies and the expected cost for any proposed system configuration. As a result, we can efficiently identify the best options without evaluating each option separately.

Chapter Twelve presents conclusions. The new interpolation and quadrature methods presented in this thesis present an opportunity to apply systems analysis to a wide variety of problems that previously were beyond the capability of DDP. A few of these problems are discussed, as well as future research to further advance GDP methods.

CHAPTER 2.

ANALYSIS OF RESERVOIR SYSTEMS

This thesis develops techniques for management of stochastic dynamic systems. These techniques are especially useful in the management of reservoir systems that regulate variable and uncertain streamflows for water supply, water quality, flood control, power generation, or other purposes. Management of such systems can include a variety of decision-making efforts, and this thesis focuses on two: the identification of effective real-time control decisions, and the evaluation of capacity expansion decisions.

Real-time control decisions (i.e., decisions that use information as it becomes available) are necessary for effective system operations. For example, we require real-time water release decisions for reservoir operations. Appropriate decisions are those that best achieve the goals of reservoir management; when failure to meet these goals is identified as a cost (e.g., of water rationing or of flood damage), then appropriate decisions are those that minimize cost.

Likewise, with increasing constraints on our use of water resources, capacity expansion decisions are essential for successful system planning. For example, we may add storage capacity to a reservoir system to reduce the likelihood of water rationing or flood damage. Appropriate decisions are those that combine development alternatives to achieve system goals in a cost-effective manner.

To identify and evaluate reservoir system operations and plans, we can apply simulation and optimization in a systems analysis approach. However, systems analysis can be difficult when we must consider the impact that uncertain inputs, such as streamflow and demand, has on the efficient regulation of systems. Uncertainty makes it impossible to precisely identify the future impact of management decisions, and the best decisions are identifiable only with hindsight. As a result, efficient regulation of stream-reservoir systems is difficult, and there is no one optimization method that we may employ in all systems analysis efforts.

This chapter presents background on systems analysis applied to reservoir systems, and illustrates why uncertain inputs make it difficult for us to apply systems analysis. To apply systems analysis to reservoir systems, we need an appropriate optimization method that can overcome these difficulties. In Chapter Four, I present one optimization method, "discrete dynamic programming" (DDP), as an appropriate method

for a variety of water resource and reservoir management problems. DDP is particularly appropriate because of its ability to represent system dynamics and constraints realistically. I present analyses of a few of these problems starting with Chapter Nine of this thesis.

In between, Chapters Five through Eight present techniques developed to improve DDP. These techniques allow us to solve more complex systems analysis problems than previously possible. Such improvements are necessary since the application of traditional DDP requires significant and sometimes excessive simplification of system models. Chapters Five and Seven present these techniques and Chapters Six and Eight evaluate their performance by application to test cases.

A. USING SIMULATION AND OPTIMIZATION IN RESERVOIR SYSTEM MANAGEMENT

To aid us in making such reservoir system operating and planning decisions, we can develop and apply mathematical models that simulate the structure and dynamics of reservoir systems. These models allow us to observe the performance of different control policies and system configurations under the influence of different streamflow scenarios.

The historical record is but one possible scenario that we may use to simulate the future. However, it is the most appealing in practice because the historical record is more concrete and less conjectural than other scenarios that we may use. Also, when the public applies hindsight to management decisions following an extreme event, such as a drought or a flood, it tends to be more critical of failures occurring with extremes that resemble historical events than of failures occurring with extremes that have not been observed previously [Glantz, 1982]. For example, though there was considerable criticism of water agencies in the western United States following the surprisingly severe drought of 1976-77, it is likely that the public's criticism was somewhat muted because such an extreme event had not previously been recorded. Now with this hindsight, a number of agencies use this drought as a basis for water rationing decisions [EBMUB, 1992; Gilbert, 1986].

Nevertheless, it is not generally appropriate to use only the historical record of streamflows (or other uncertain input) to test management decisions. It is unlikely that the future will repeat the pattern of the past. In addition, the past may not be an accurate guide to the future because of non-stationary processes that contribute to hydrologic conditions. The impact of changing land-use and weather patterns may significantly alter the magnitude and pattern of future rainfall and runoff. Even when conditions are stationary, the historical record provides but one possible scenario of past streamflows

(when viewed in a probabilistic sense) that is not representative of the range of possible future flows [*Fiering and Jackson, 1971*].

Instead of relying only on the historical record, we may use synthetic streamflow models to generate a number of random scenarios for river basin planning [*Fiering and Jackson, 1971; Loucks et al., 1981; Salas et al., 1980*]. We then attempt to make the best decisions possible after consideration of the possible outcomes and their likelihood. When faced with a large number of potential outcomes however, we find it difficult to identify the most appropriate management decisions. To aid our search effort, we can apply an optimization method that automatically identifies the "best" decisions. Thus, we may identify the best management options more quickly and efficiently by using a systems analysis approach than by using only simulation.

B. THE IMPACT OF STOCHASTIC INPUTS

The reason most reservoir systems exist is to moderate the variability and uncertainty of streamflows. Streamflows are variable because they change with transient patterns of precipitation that also vary with season and perhaps long-term trends reflecting climate change. In addition, streamflows are uncertain because the variable pattern of flows cannot be identified in advance (note that streamflows cannot be uncertain unless they also are variable). We say that streamflows are "stochastic" because they contain an erratic component that makes precise prediction of future flows impossible. Reservoir system conditions change in response to stochastic streamflow inputs in ways that are not completely within our control, and, as a result, management of a reservoir system presents what we call a "stochastic dynamic control" problem.

A major challenge in the management of reservoir systems is how to anticipate and regulate stochastic streamflow inputs that drive system dynamics. The uncertainty of inputs is large, and we cannot identify efficient controls far in advance of actual operations. Instead, we make control decisions only when needed, thereby allowing us to collect as much information as we can before committing ourselves to a particular course of action. For example, when managing a flood-control reservoir, we make release decisions just prior to making actual releases so that we may take advantage of the most up-to-date streamflow forecasts. Such decisions are examples of "real-time control."

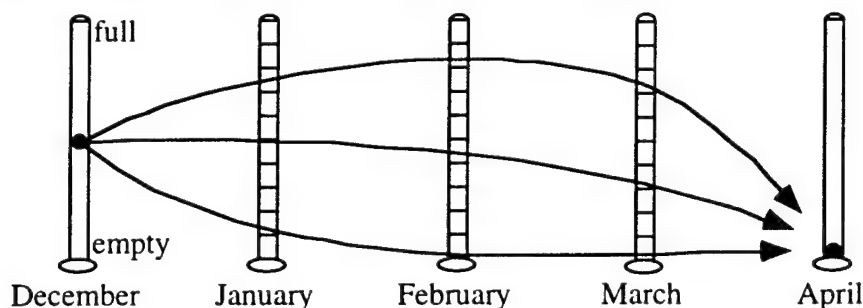
1. An Example of Management Without Uncertain Inputs

It is easier to regulate a system that does not have streamflows or other inputs that are stochastic. In such cases, we know everything, in advance, about the system and

there is no advantage in delaying control decisions. We say that such a system is “deterministic” because all information about the system is (or is assumed to be) determined precisely. As a result, we can identify, in advance, a single “best” schedule of control decisions for the entire operating horizon. So long as our information about the system doesn't change, we can use these controls to forecast the evolution of the system.

As a simple example, suppose we need to regulate the level of a reservoir to meet water supply and flood control needs. If future inflows are determined precisely, we can identify the single best schedule of releases and the associated trajectory that describes the evolution of reservoir levels. Figure 2B1 illustrates possible trajectories that can take us from a half-full reservoir in December (the “current” month) to an empty reservoir in April, required perhaps to prevent flooding in May and June. The best trajectory is that associated with a release schedule that respects constraints on system operation, and that minimizes costs. These costs may quantify the impacts of flooding, water use, or other purposes.

Figure 2B1. Example Trajectories for Regulation of Reservoir Level



2. The Effect of Uncertain Inputs on Reservoir Control

In contrast, system regulation is difficult if a reservoir system has uncertain inputs. Uncertain inflows, for example, make it impossible to identify both future release decisions and reservoir levels. Instead of identifying a single schedule of control decisions, we adopt a “wait-and-see” approach to delay decisions as long as possible. By delaying, we can observe the values of some streamflows and may be able to reduce the uncertainty of other future flows. Using this information, we can identify releases that better achieve system management goals. Nevertheless, we can identify the truly best schedule of releases only with hindsight.

To continue with the simple example, suppose an unexpected storm occurs in February. If we had identified, in advance, a single control schedule that is insufficiently cautious, then the storm could cause flooding. Figure 2B2 illustrates the effect that such

an unexpected event might have on a planned trajectory. In this illustration, the storm causes flooding because the reservoir fills and spills.

With hindsight, we can easily identify that a more cautious release schedule would have prevented this flood; however, we cannot wait for hindsight. Instead, we must make release decisions in advance. At best, we can make release decisions in real-time by delaying each release decision until the month it is required. By delaying decisions, we can observe prior months' streamflows and the resulting reservoir levels, and perhaps we may also reduce the uncertainty of future flows. For example, we may not be able to foresee a February storm in December, but perhaps we may be able to anticipate the storm in January. By delaying control decisions, we can reduce the flood damage by adjusting January's control decisions to reflect this additional information (Figure 2B3). This is an example of real-time control because we adjust releases in real time using information from changing conditions and forecasts.

In addition, we may prefer a more conservative control path, regardless of whether we can improve predictions by delaying control decisions. If the impact of infrequent flooding is unacceptably severe, we may drain the reservoir more rapidly to make additional storage available to capture flood waters (Figure 2B4). We refer to this preference as "caution." Cautious decisions hedge against the occurrence of extreme events that are potentially catastrophic, even though they are rare. Under ordinary streamflow conditions, cautious decisions may seem unduly conservative; however, in the long run, appropriately cautious decisions should be more effective.

Figure 2B2. Effect of Unexpected Storm on Regulation

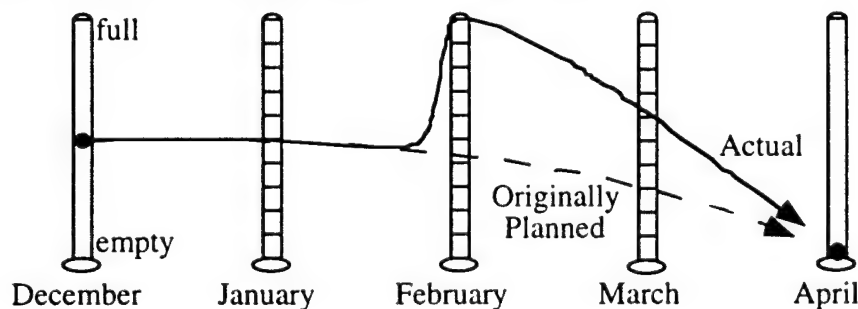


Figure 2B3. Effect of Real-Time Control and Better Foresight on Regulation

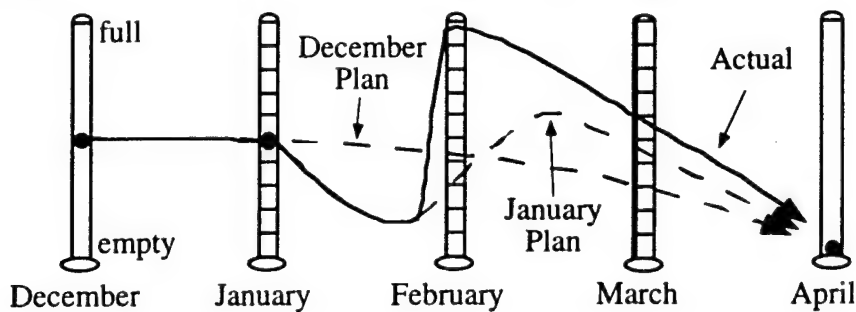
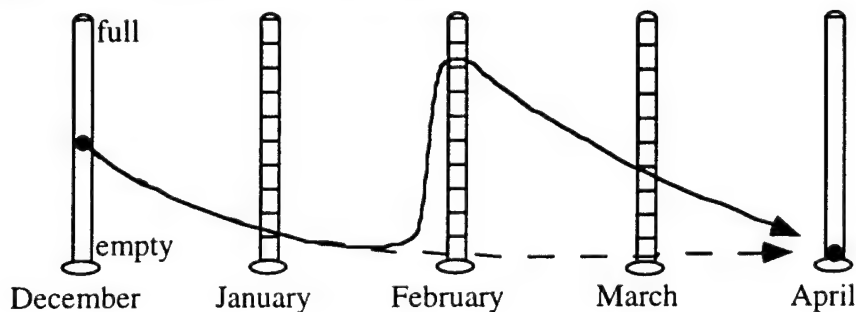


Figure 2B4. Effect of Caution on Regulation



3. The Purpose of Feedback Control

For the simple reservoir example with stochastic inflows, we see that we cannot simultaneously identify a schedule of release decisions and future reservoir levels. If we specify release decisions in advance, then reservoir levels will vary with the cumulative impact of inflow variability. If we specify reservoir levels in advance, then release decisions must vary with inflows to achieve target levels.

Using real-time control, we do not specify either decisions or levels in advance. Instead, we use a “wait-and-see” approach to delay decisions as long as possible. This allows us to observe the impact of inflows on reservoir levels and to reduce the uncertainty of future inflows. With this information, we are better able to choose decisions and levels that produce the best “expected” system performance (i.e., lowest expected cost) averaged over possible future scenarios. In other words, we “feed” this information back into the decision making process to improve control decisions.

Whenever stochastic inputs have a cumulative impact on system characteristics (i.e., in any real-time control problem), we must provide a mechanism for using this information to update control decisions. This is known as “feedback control.” When using systems analysis, the mechanism for feedback control depends on the optimization method that we use to update control decisions. In this thesis, I use the method of

discrete dynamic programming (DDP) for reasons that we will discuss in Chapter Four. In addition, Chapter Three will briefly consider other optimization methods that may be used for feedback control.

4. The Impact of Reducible and Irreducible Uncertainty

For purposes of systems analysis, it is useful to distinguish between uncertainty that is reducible and uncertainty that is irreducible. As we have discussed, if we can reduce the uncertainty of inputs as system operations progress, then we can improve system control by delaying decisions. This uncertainty is dynamic and reducible, and we should incorporate it in a system model as a stochastic input. In contrast, if we cannot reduce the uncertainty of inputs as operations progress, then there is no benefit from delaying decisions. We may refer to this uncertainty as irreducible, and we may incorporate this in system models without need for real-time control.

An input may have irreducible uncertainty if we cannot observe its impact on system dynamics. For example, groundwater models typically contain uncertain parameters for hydraulic conductivity and storativity. Because it is impractical to directly obtain values for many of these parameters, we infer parameter values from indirect tests conducted prior to modeling. Using our best estimates of these values, we create models and apply systems analysis using deterministic optimization. In these cases, we can test the impact of uncertainty on results by generating a number of Monte Carlo realizations. This allows us to test the range of possible outcomes, but we still can apply deterministic optimization to each realization.

Often, we view uncertainty as irreducible to make a problem easier to solve. In truth, there are few uncertain inputs that have a completely unobservable impact on system dynamics. For example, we may observe the response of groundwater flow to control decisions. As recognized by several authors [*Georgakakos and Vlatas*, 1991; *Lee and Kitanidis*, 1991; *Whiffen and Shoemaker*, 1993], we can use this response to improve parameter estimates and reduce uncertainty in groundwater models. Deliberate selection of control decisions to improve parameter estimates is known as “probing” [*Lee and Kitanidis*, 1991].

Though reducible and irreducible uncertainty both affect optimal control decisions, they have different representations in system models. For clarity, this thesis ignores the effect that irreducible uncertainty can have on reservoir management. In particular, as an example of irreducible uncertainty, this thesis will ignore the effect that measurement uncertainty has on optimal control. For discussion of how we may

incorporate irreducible uncertainty into reservoir management, one may view *Stengel* [1994] or other works on optimal control.

C. DEVELOPING SYSTEMS ANALYSIS FRAMEWORK

Systems analysis involves (1) the development of a mathematical model that simulates the essential elements of an actual system, (2) the development of a value model that identifies management goals and that quantifies system performance, and (3) an optimization procedure that identifies management policies that achieve the best system performance.

A system is a group of components meant to perform a common function. Reservoir systems consist of interconnected storage components that function to regulate variable inputs such as streamflow or demands for water supply and power generation. We can judge the effectiveness of system management by measuring system performance against various criteria such as maintaining prescribed reservoir levels or meeting demands for water. Table 2C1 identifies some goals of stream-reservoir system management and some possible performance measures.

We need not limit system management to a single goal, and "multi-purpose" systems are common. For example, a reservoir constructed for water supply may also be used for power generation. We may assess the performance of multi-purpose systems by a single quantitative measure or through a "multi-objective" analysis. Multi-objective analysis compares tradeoffs between objectives without identifying the relative value of each purpose explicitly [*Major*, 1977]. This thesis will not consider the problem of assessing relative values of system performance for each objective, but will assume that a single performance measure exists.

Associated with the goals and performance measures of Table 2C1, we can identify control decisions and information that may aid selection of appropriate controls. Identification of appropriate controls often requires consideration of information that defines the Markovian "state" of a system. We can identify this state information by a set of variables summarizing what we know about the present condition of a system. Generally, the values of these state variables will change with time, subject to applied controls and stochastic inputs.

Table 2C1. Common Measures of Performance, Control, and State

Goal	Performance Measure	Control	State
Water Supply	Cost of water rationing	Allocation	Amount stored Long-term flow forecast
Hydropower	Revenue from power generation	Turbine operation	Power head Short-term flow forecast
Flood Control	Cost of flood damage	Release	Free storage Short-term flow forecast
Fish Habitat	Fish population	Release	River stage Water temperature
Recreation	Deviation from desired levels	Release	Reservoir level River stage

1. Mathematical Model of a System

A system model is a concise mathematical description that identifies controllable inputs, the state of facilities and other system components, uncontrolled and uncertain inputs, system dynamics, and constraints.

DECISION VARIABLES

Regulation of a system requires the existence of system inputs that are controllable and that can modify system dynamics. For stream-reservoir systems, the most common and obvious inputs under our control are reservoir releases. In addition, we may also control allocation of supplies downstream and other types of flows within a system. We (i.e., the manager or operator of a system) regulate controllable system inputs by control decisions that we make. We represent these decisions by variables $\mathbf{u} = [u_1, u_2, u_3, \dots]^T$ arranged in a column vector.

STATE VARIABLES

Effective regulation of a system requires appropriate use of information about the condition of the system. For stream-reservoir systems, this information will commonly include characterization of reservoir levels, streamflow forecasts (including related information such as current streamflow, snowpack measurements, soil moisture, weather forecasts, etc.), demand forecasts, status of equipment and facilities, and so forth. This

information defines the state of a system that we represent as n variables $\mathbf{x} = [x_1, \dots, x_n]^T$ arranged in a column vector.

Because control decisions depend on a system's state, we can specify a functional relationship $\mathbf{u}_{(t)} = \mathbf{U}_t(\mathbf{x}_{(t)})$ describing a course of action based on the system's state at time t . This relationship describes the "control policy" of a system. For example, we can specify optimal water releases from a reservoir as a function of current reservoir levels, streamflow forecasts, season, or other state information. We can also summarize such control policies by "rule curves" that prescribe control decisions for system operators [Marien et al., 1994].

STOCHASTIC VARIABLES

Often, a variety of uncertain and uncontrollable inputs affect the evolution of a system. For example, stochastic streamflows effect the evolution of reservoir levels and impact control decisions. We represent these stochastic inputs as variables $\mathbf{s} = [s_1, s_2, s_3, \dots]^T$ arranged in a column vector.

Part of our modeling effort is to identify functions that describe of the probability distributions of these inputs. I use a function $s(\mathbf{w})$ to describe a transformation of m random normal variables $\mathbf{w} = [w_1, \dots, w_m]^T$ to the stochastic variable s . For example, if streamflows are lognormally distributed, we may use the model

$$s = \exp[\mu + \sigma w]$$

to describe the transformation of random variable w to stochastic variable s . In this example, parameters μ and σ are the mean and standard deviation of the log-transformed sample values of s .

Such a functional description can also depend on the state of the system. For example, if streamflows are autocorrelated, we may use the model

$$s = \rho x + \exp[\mu + \sigma w]$$

Parameter ρ is a correlation coefficient and $x_{(t)} = s_{(t-1)}$ is a state variable representing the prior period's flow. When modeling a stochastic input, we can use any appropriate function $s(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)$ of state variables, decision variables, random variables, or time, though we should prefer simple and clear models. In practice, we use random normal variables \mathbf{w} to represent uncertainty, and we incorporate functional descriptions of stochastic variables s implicitly in other model equations.

DYNAMICS

The state of a system evolves in response to the application of controlled and uncontrolled, uncertain inputs. We can model this dynamic behavior by a vector first-order differential equation that describes a continuous change of state \mathbf{x} :

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)$$

The evolution of state \mathbf{x} depends on its current value, on inputs \mathbf{u} and \mathbf{w} , and on the current stage.

If we update controls periodically instead of continuously, it is easier to describe the system's evolution for discrete values of time $t_j = t_1, \dots, t_N$. These values break a control problem into "stages" (i.e., time intervals) of length Δt_j , where $t_{j+1} = t_j + \Delta t_j$. For each stage, we can model a system's dynamic behavior by discrete changes

$$\Delta \mathbf{x}_{(t)} = \int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t) dt$$

In addition, at any discrete time t_j , we can identify the state of a system by a function

$$\mathbf{x}_{(t_{j+1})} = \mathbf{T}_{t_j}(\mathbf{x}_{(t_j)}, \mathbf{u}_{(t_j)}, \mathbf{w}_{(t_j)})$$

where $\mathbf{x}_{(t_j)}$ and $\mathbf{x}_{(t_{j+1})}$ are the states at the beginning and end of stage t_j , and $\mathbf{u}_{(t_j)}$ and $\mathbf{w}_{(t_j)}$ are controlled and uncontrolled inputs applied during the stage. This equation identifies the "transition function" of a dynamic system. To avoid subscripts, I will frequently use the shorthand definitions

$$\mathbf{y} \equiv \mathbf{x}_{(t_{j+1})} \quad , \quad \mathbf{x} \equiv \mathbf{x}_{(t_j)} \quad , \quad \mathbf{u} \equiv \mathbf{u}_{(t_j)} \quad , \quad \mathbf{w} \equiv \mathbf{w}_{(t_j)} \quad , \quad t \equiv t_j \quad (2C1)$$

Using these definition, we identify the transition function by

$$\mathbf{y} = \mathbf{T}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2C2)$$

Similarly, we identify the control policy and stochastic models by

$$\mathbf{u} = \mathbf{U}_t(\mathbf{x})$$

$$\mathbf{s} = \mathbf{S}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2C3)$$

CONSTRAINTS

A system model should not allow us to select infeasible control decisions. For example, reservoir releases cannot be negative. Feasibility may also depend on non-physical constraints that incorporate public policy considerations, contractual obligations,

or other legal constraints. Sometimes however, we may neglect certain feasibility constraints because they have a negligible impact on control policies (i.e., constraints that are never binding, or at least not during critical periods of system operation). For example, storage capacity in a reservoir system may be sufficiently large to have a negligible impact on limiting allocation decisions.

We can represent inequality constraints in general form as

$$\text{lower bound} \leq f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \leq \text{upper bound}$$

where f is any function of the variables \mathbf{x} , \mathbf{u} , and \mathbf{w} (or \mathbf{s}). For equality constraints, the lower and upper bounds are equal. Under certain conditions, we can re-express the set of constraints as

$$\mathbf{B}_t^L(\mathbf{x}, \mathbf{w}) \leq \mathbf{u}_{(t)} \leq \mathbf{B}_t^U(\mathbf{x}, \mathbf{w}) \quad (2C4)$$

where \mathbf{B}^L and \mathbf{B}^U are arrays of functions that define, respectively, the lower and upper bounds of control variables. Subscripts on these arrays indicate that constraints may depend on the current stage t .

With stochastic inputs, constraints on decisions are also stochastic. When the impact of \mathbf{w} on bounds is small, we can choose decisions that are feasible for all reasonable values of \mathbf{w} . However, when the impact of \mathbf{w} is large, control decisions may be unreasonably constrained or may have no values feasible for all \mathbf{w} .

To avoid stochastic constraints that unreasonably restrict available control decisions, we can make constraints deterministic by assuming foresight of \mathbf{w} for the current stage. This also reduces the work for each optimization to identify the best control decisions. However, we now identify control decisions by a policy that is a function of both the initial state and random variables for the current stage:

$$\mathbf{u}_{(t)} = \mathbf{U}_t(\mathbf{x}, \mathbf{w}) \quad (2C5)$$

As a result, we must identify optimal decisions for each possible outcome of \mathbf{w} that we use to represent the stochastic distribution. As we will see, this can greatly increase the number of optimizations that we must perform, balancing out any reduction in work for each optimization.

FEASIBILITY OF CONTROL DECISIONS WITH STOCHASTIC CONSTRAINTS

In many hydrologic systems, the uncertainty of hydrologic variables can be quite large, making it difficult to fix control decisions in advance, even for the current stage. Frequent "tweaking" of control decisions is often required to adjust for changing hydrologic values. When developing mathematical models, we would like, ideally, to use

stage lengths that are small enough to allow tweaking of controls in a manner similar to actual operations. However, this may impose an unrealistically large computational burden that requires use of longer stages.

Three approaches are available to ensure feasibility of control decisions under conditions of uncertainty. These are to (1) identify fixed controls that are feasible for all discrete outcomes, (2) identify controls that use some foresight of stochastic inputs, and (3) tolerate limited occurrence of infeasible controls as a form of model error. Each of these approaches requires simplification of the system model, and practical application may require tweaking of the identified controls.

When using long stages, all the above approaches result in solution error. Because the first approach requires fixed controls that are feasible for all discrete outcomes, the range of available controls can be significantly constrained. This results in controls that are excessively conservative (i.e., cautious) and cost more than expected from actual operations. Because the second approach develops control decisions using prior knowledge of streamflows in the current stage, solutions may have an unrealistic advantage. This results in controls that are less conservative and cost less than expected from actual operations.

The systems analysis methods presented in this thesis ensure feasibility of control decisions by using some foresight of stochastic inputs. Specifically, these problems assume foresight of current stochastic inputs s (or w) when identifying control decisions u . However, this foresight is "limited," because we use it only to identify current control decisions and not to identify the cost of current and future operations. In practical application, the resulting control decisions can be tweaked as observations of the true outcome are available.

ARGUMENTS FOR USING LIMITED FORESIGHT

Under most conditions, the use of limited foresight should produce better control decisions than alternative approaches. In particular, it is appropriate to use limited foresight when real-life changes in regulation occur more rapidly than simulated by a model. On the other hand, when the time required to implement changes is longer than the length of stages used in a model, it may be more appropriate to identify fixed controls for all outcomes of current stochastic inputs. However, there is little advantage to using stages that are shorter than the time required to implement changes. Moreover, using numerous, short stages can impose a significant computational burden.

The following is a list of practical reasons that justify the use of foresight:

(1) Feasibility of Control Decisions

Not using foresight may produce stochastic constraints for which there are no feasible decisions or for which the set of feasible decisions is unrealistically small. This is especially true when long stages are required to reduce the computational burden.

(2) Limited Computer Resources

Limited computer resources often require us to use long time intervals in system models. Real-time control of the true system will often be at shorter intervals, so the assumption of foresight produces a control policy that is closer to what we should use in practice. Also, with long time intervals, the impact of w is large and we may produce an empty set of feasible decisions as in (1).

(3) Convergence of Solutions with Short Stages

Even when we have the luxury of using short time intervals to reduce the impact of w on stochastic constraints, there is little benefit in not using foresight. As the impact of w diminishes, the significance of assuming foresight also diminishes. As a result, there will usually be a negligible difference between control policies that use foresight and those that do not.

(4) Unboundedness

A final—and perhaps most compelling—reason for not requiring feasibility for all reasonable w is that identification of reasonable w is arbitrary: do we bound reasonable w using a hundred-year or thousand-year flood event? If we use wide bounds to model extreme but rare events, then the impact of w may again become large enough to excessively constrain control decisions.

2. Value Model of a System

It is meaningless to talk about managing a system effectively without explicitly stating a measure of system performance. We require such a measure to compare the performance of one control policy or system-expansion plan with any other. For example, we may judge performance by how well inputs are regulated to minimize deviations from desired levels or to maximize some output. To measure system performance, we must develop a value model that explicitly quantifies this performance.

Value models are often subjective and derived from a variety of decision makers with divergent views. As a result, identification of a single value model can be contentious [Rogers and Fiering, 1986]. Even when we measure performance by indices

such as minimizing absolute deviations or minimizing the square of deviations, these indices contain implicit judgments about the cost of deviations.

When regulating a multi-objective system, it can be particularly difficult to identify a value model. Multi-objective systems require that we balance the costs and benefits of different objectives that may be in conflict and that may not have a common basis for measurement. Unfortunately, most reservoir systems require multi-objective analysis: Even when a system is designed for a single-purpose, multiple objectives exist because resources are limited and must be balanced with achieving the purposes of a system.

ACCURACY OF VALUE MODELS

Though it may be difficult to provide a precise measure of system performance, an approximate performance measure is better than none. An approximate measure can still incorporate significant information that will result in effective system management. For example, though we may not have a precise measure for the value of water supplies, we do know that the value of additional water generally diminishes with increasing supply. We can incorporate this information into an approximate value model, and better management will result than from management without any explicit or implicit value model.

If we have some prior knowledge about appropriate management solutions, we can also use this to identify a value model. By using an iterative approach, we can progressively define a satisfactory value model. If an initial model produces an unsatisfactory solution, we can modify the model until we attain an appropriate solution. Using this approach, we can transform soft judgments and preferences into quantified performance measures. This also suggests that we can assess the implicit value judgments used in pre-determined management solutions. Likewise, we can evaluate the cost of constraints on operating and planning decisions.

In this thesis, I use a variety of value models. Chapters Six, Eight and Nine use abstract value models used previously by other authors. Chapters Ten and Eleven use models based on reasonable values for water supply and for the cost of system management. Though reasonable, these value models still contain significant uncertainty. Various authors have worked at reducing uncertainty of water supply values [*Cameron and Wright*, 1990; *Martin and Kulakowski*, 1991; *Martin and Thomas*, 1986; *Williams and Suh*, 1986; *Young*, 1973], and Chapter Ten summarizes some of the results that we can use to provide values for the management of urban water supplies.

VALUE FUNCTION DESCRIPTION

We can represent a value model in general form as

$$\frac{dV}{dt} = \dot{V}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)$$

This allows us to incorporate costs associated with the system state, controls, stochastic inputs, and time. If we update controls periodically instead of continuously, we can evaluate a “cost” for stage t as

$$\Delta V = \int_t^{t+\Delta t} \dot{V}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t) dt$$

which we can re-express as a function

$$\Delta V = C_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2C6)$$

where C_t is a “current” cost associated with stage t .

By evaluating the cost for each stage, we can assess the total cost (and “performance”) for all stages by the summation

$$V_{t_1} = \sum_{t=t_1}^{t_N} C_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{N+1}}(\mathbf{x}) \quad (2C7)$$

The function $F_{t_{N+1}}$ represents the long-term “future” cost or preference associated with a final state $\mathbf{x}_{(t_{N+1})}$. For example, if we do not care about the impact of this final state on costs after stage t_N , we can set $F_{t_{N+1}}(\mathbf{x}) = 0$ for all $\mathbf{x}_{(t_{N+1})}$. On the other hand, if we need to attain some specific final state, either we can specify $\mathbf{x}_{(t_{N+1})} = \mathbf{x}^{(i)}$ as a constraint, or we can specify $F_{t_{N+1}}(\mathbf{x}) = 0$ for $\mathbf{x}_{(t_{N+1})} = \mathbf{x}^{(i)}$ and a penalty cost $F_{t_{N+1}}(\mathbf{x}) > 0$ for deviations from the target.

In this thesis, I refer to the values produced by such models as “costs.” This implies that the best operations and plans are those that “minimize” the cost from a value function. We can incorporate revenues or other benefits as negative costs.

PERFORMANCE IN THE PRESENCE OF UNCERTAIN INPUTS

We cannot precisely evaluate the total cost by equation (2C7) when some system inputs are uncertain. As discussed earlier, we cannot identify both control decisions and future states. We can identify actual system performance only after we have an opportunity to observe actual values of all inputs (i.e., with hindsight).

Instead, because we must make control decisions in real-time, we can identify decisions that achieve the best possible “expected” performance. Expected performance

weighs possible outcomes of uncertain future inputs. For each control policy or schedule of proposed controls, we calculate cost as a weighted average of all possible outcomes. For example, if there are M possible outcomes of \mathbf{w} in stage t_j , the expected single-stage cost is given by the sum

$$E_{\mathbf{w}}\{ \Delta V \} = \sum_{k=1}^M v_k C(\mathbf{x}, \mathbf{u}, \mathbf{w}^{(k)}) \quad (2C8)$$

Each parameter v_k is the weight applied to the k 'th outcome $\mathbf{w}^{(k)}$ of the random variables.

When a random variable has a continuous distribution, there is an infinite number of possible outcomes. As a result, the single-stage cost of equation (2C8) should be evaluated by the integral

$$E_{\mathbf{w}}\{ \Delta V \} = \int_{-\infty}^{+\infty} W(\mathbf{w}) \{ C(\mathbf{x}, \mathbf{u}, \mathbf{w}) \} d\mathbf{w} \quad (2C9)$$

Function $W(\mathbf{w})$ is the probability density of \mathbf{w} used to weight the different outcomes. When \mathbf{w} contains normally-distributed random variables, $W(\mathbf{w})$ has a multivariate Gaussian distribution [Keeping, 1995]. However, analytic evaluation of integrals such as in equation (2C9) is usually difficult.

Instead, we approximate expected-cost integrals using numerical integration, or "quadrature" using summations similar to equation (2C9) [Press et al., 1992]. To apply quadrature to equation (2C9), we need to identify a number of discrete outcomes $\mathbf{w}^{(k)}$ (or "abscissas") and appropriate weights v_k . Abscissas are typically equally spaced outcomes at the nodes of a grid spanning the random variables. Weights depend on the abscissas, and there are several appropriate methods that we can use to evaluate these weights. For example, we may use the probability density function to define weights given by:

$$v_k = \prod_{j=1}^J \{ \Delta w_j W(w_j^{(k)}) \} \quad (2C10)$$

This approximation is known as the trapezoidal rule [Press et al., 1992]. There are other, more sophisticated, numerical integration methods that we will consider in Chapter Seven.

3. Optimization of System Performance

Our goal is to regulate a system to achieve the best possible performance. Given a system and value models for a system, our goal is to identify management decisions

that minimize system costs over a time horizon of interest. To identify these decisions, we need to employ some search method. A search method that automatically identifies the least-cost, or “optimal,” decisions is also known as an “optimization” method.

Depending on the characteristics of a reservoir management problem, we may choose to employ one of a variety of optimization methods. Problems considered in this dissertation are appropriately addressed by relatively simple “lumped-parameter” models because we can describe important state information—such as reservoir levels, streamflow forecasts, etc.—by a few state variables. Also, as discussed earlier in this chapter, reservoir management requires a cautious optimization method that can reduce the cost of extreme events. Because of these characteristics, I have relied on the optimization method of DDP discussed in Chapter Four.

It is difficult to solve control problems with stochastic inputs, and all optimization methods rely on special characteristics of a problem to make it tractable. To help explain why it is difficult to solve stochastic management problems, the remainder of this chapter outlines the brute-force approach to finding real-time control decisions.

DIFFICULTY IN IDENTIFYING PERFORMANCE FOR ALL STAGES

With no stochastic inputs, we can identify a single optimal control schedule and the resulting system performance in advance. For a discrete initial state, $\mathbf{x}_{(t_1)}^{(i)}$, we can evaluate the optimal schedule of control decisions $\mathbf{u}_{(t_1)}^*, \dots, \mathbf{u}_{(t_N)}^*$ and resulting system performance V^* by

$$V_{t_1}^*(\mathbf{x}^{(i)}) = \min_{\mathbf{u}_{(t_1)}, \dots, \mathbf{u}_{(t_N)}} \{ V(\mathbf{x}_{(t_1)}^{(i)}, \mathbf{x}_{(t_2)}, \dots, \mathbf{x}_{(t_{N+1})}, \mathbf{u}_{(t_1)}, \dots, \mathbf{u}_{(t_N)}) \}$$

As before, future states are defined by transition functions and constraints. Optimal control decisions and the resulting system performance can be identified relatively easily using an appropriate deterministic optimization method.

With stochastic inputs, we cannot—in advance—identify a single optimal control schedule or the exact system performance. Using feedback control, there are an infinite number of possible control schedules, and we can only identify the best with hindsight. Also, a value model quantifies system performance for *specific* inputs, so we cannot evaluate actual performance until we know all inputs.

Instead, we identify a set of policies, one for each stage, that provide control decisions for any possible outcome of the stochastic inputs. System performance is evaluated as the *expected* performance averaged over all possible outcomes with consistent application of the control policies. As discussed before, policies are functions

of the current state \mathbf{x} that we observe and, when using limited foresight, of the current random variables \mathbf{w} .

The expected performance of a series of control policies $\{U_{(t_1)}, \dots, U_{(t_N)}\}$ for N stages is given by

$$E\{V\} = E_{\mathbf{w}_{(t_1)}, \dots, \mathbf{w}_{(t_N)}}\{V | U_{(t_1)}, \dots, U_{(t_N)}\} \quad (2C11)$$

and the series of optimal control policies $\{U_{(t_1)}^*, \dots, U_{(t_N)}^*\}$ is given by

$$F^* = \min_{U_{(t_1)}, \dots, U_{(t_N)}}\{E\{V\}\} \quad (2C12)$$

where F^* is the optimal future performance expected under these policies.

In contrast to the deterministic case with no stochastic inputs, optimal control policies and the expected system performance are difficult to identify. This is particularly true if we use equations (2C11) and (2C12) directly. The expected value of equation (2C11) is the multiple integration

$$E\{V\} = \int_{-\infty}^{+\infty} W(\mathbf{w}_{(t_1)}) \dots \int_{-\infty}^{+\infty} W(\mathbf{w}_{(t_N)}) \{V\} d\mathbf{w}_{(t_N)} \dots d\mathbf{w}_{(t_1)} \quad (2C13)$$

for a known series of control policies. Analytic solution of this equation is difficult, so we apply numerical integration analogous to equation (2C8):

$$E_{\mathbf{w}_{(t_1)}, \dots, \mathbf{w}_{(t_N)}}\{V\} \approx \sum_{k_1=1}^M v_{t_1, k_1} \dots \sum_{k_N=1}^M v_{t_N, k_N} \{V\} \quad (2C14)$$

To evaluate equation (2C14) once for an initial state $\mathbf{x}_{(t_1)}^{(i)}$ and one series of control policies $\{U_{(t_1)}, \dots, U_{(t_N)}\}$ requires considerable effort: With M combinations of \mathbf{w} for each stage, there are M^N possible outcomes, each requiring an evaluation of the total cost given by $V(\mathbf{x}_{(t_1)}^{(i)}, \mathbf{x}_{(t_2)}, \dots, \mathbf{x}_{(t_{N+1})}, \mathbf{u}_{(t_1)}, \dots, \mathbf{u}_{(t_N)}, \mathbf{w}_{(t_1)}, \dots, \mathbf{w}_{(t_N)})$. In addition, we must evaluate N transition functions and associated constraints. Suppose we evaluate a simple ten-stage model with two random variables. If we consider three possible outcomes for each random variable, the number of possible outcomes in each stage is $M = 3^2$, and the total number over ten stages is $M^N = (3^2)^{10}$, or about 3.5 billion. This is hardly a modest effort! As we see, the number of scenarios is essentially infinite for most realistic problems.

For any stochastic control problem but the most trivial, it is futile to use equation (2C11) to evaluate the total cost function and equation (2C12) to search for optimal policies. Instead, we must identify a less general formulation and systems-analysis method. This formulation must take advantage of problem-specific characteristics and

appropriate simplifying assumptions. In the absence of appropriate simplifying assumptions, the formulation must employ assumptions judged to be least detrimental to the accuracy and validity of resulting management decisions. As we will see in Chapters Three and Four, one formulation involves breaking the control problem into smaller, easier-to-solve sub-problems using one of several dynamic programming methods.

DESCRIPTION OF SYSTEM PERFORMANCE

A function that describes the expected value of V at the time when controls are selected is also known as a "cost-to-go function" or a "future-cost function." For convenience, we may identify the future cost function by F_{t_1} where

$$F_{t_1}(\mathbf{x}) = E_{\mathbf{w}_{(t_1)}, \dots, \mathbf{w}_{(t_N)}} \{ V | \mathbf{U}_{(t_1)}, \dots, \mathbf{U}_{(t_N)} \} \quad (2C15)$$

This function specifies the expected system performance, or cost, to go from an arbitrary initial state $\mathbf{x}_{(t_1)}$ to a probabilistic final state $\mathbf{x}_{(t_{N+1})}$ at the desired time horizon. We wish to identify optimal control policies $\{\mathbf{U}_{(t_1)}^*, \dots, \mathbf{U}_{(t_N)}^*\}$ that we can use in real time to identify control decisions

$$\mathbf{u}^* = \mathbf{U}_{(t)}^*(\mathbf{x}, \mathbf{w}) \quad (2C16)$$

Optimal control policies are those that minimize the total cost function for any initial state of the system. The function $F_{t_1}(\mathbf{x})$ depends only of the system's initial state because everything else is determined: control decisions \mathbf{u} are defined by control policies; random variables \mathbf{w} are incorporated as a weighted average; and subsequent states $\mathbf{x}_{(t_2)}, \dots, \mathbf{x}_{(t_{N+1})}$ are defined by transition functions.

This cost-to-go function summarizes all information that we need to assess future system performance and to make appropriate control decisions. When solved with sufficient accuracy, this function summarizes expected future costs of system operations and appropriately weighs the cost and risk of extreme events. Also, if we know the cost-to-go function for the end of our current stage, we can solve a single-stage optimization problem to identify current controls that appropriately balance current and future costs. For example, knowledge of the future water-supply value of water stored in a reservoir allows us to identify release decisions that balance the cost of current rationing with the cost of expected future rationing. Chapter Three will show that development of a cost-to-go function in each stage is the key that allows us to break dynamic control problems into easier-to-solve sub-problems.

CHAPTER 3.

REVIEW OF OPTIMIZATION METHODS FOR STOCHASTIC DYNAMIC CONTROL

Given a system and criteria for measuring the performance of the system (such as cost), we require a systems analysis method to determine the best real-time control policy where the “best” is defined as that policy that minimizes the total expected cost of system operations up to some time horizon as modeled by equation (2C7). As we saw at the end of Chapter Two, the effort required to identify such policies could be overwhelming, and—as we might expect—there currently appears to be no general optimization method that we can apply to solve all stochastic control problems. Instead, for each type of control problem, it appears that we must identify an optimization method that takes advantage of specific problem characteristics and, perhaps, appropriate simplifying assumptions.

This chapter will present a brief summary of methods that we can use to solve stochastic dynamic control problems. As identified by *Lamond and Boukhtouta* [1995, p 32], no single method can solve all problems that have the following characteristics:

- (1) a large amount of data required to describe the state of a system
- (2) uncertain parameters that are correlated and not normally distributed
- (3) non-linear dynamics, complex objectives, and constraints

The weakness of various methods in one or more of these areas requires us to make simplifying assumptions in order to apply them, and these assumptions may be inappropriate for many problems. However, for a variety of reservoir management problems (such as those presented in Chapters Nine and Eleven), we will see that discrete dynamic programming (DDP) can be an appropriate method.

A. CURRENT STATUS OF SYSTEMS ANALYSIS APPLIED TO WATER-SUPPLY MANAGEMENT

The operators of reservoir systems have continued to rely on simulation as the principal means for deciding what control policy to use. The use of optimization methods has been limited, primarily because of past difficulties in applying systems-analysis to realistic system models [Rogers and Fiering, 1986], and partly because of difficulties in quantifying the measures of system performance used in reservoir management [Brookshire *et al.*, 1986].

The exception to these general observations appears to be in the application to systems that have hydropower generation as the sole (or principal) goal of operations. In hydropower systems, it is easy to compare the benefits attained under different policies. Also, the simplifying assumptions have had only a reasonably small effect on the accuracy and appropriateness of the resulting control policies. Policies obtained using systems analysis have been successful in achieving larger hydropower benefits than well-established policies obtained using efficient heuristic methods. In hydropower systems, marginal improvements can be quantified, and systems-analysis methods have produced benefits amounting to millions of dollars [Kelman *et al.*, 1990]. Systems analysis methods have been applied to other types of systems to a lesser extent. The models and assumptions used in hydropower management have not been found appropriate for other types of reservoir problems.

In this thesis, systems analysis methods are developed for application to water-supply management. This application has been motivated by my interest in ways to efficiently manage systems that use both surface water and groundwater (i.e., "conjunctive use") for urban supply. The structure of water-supply systems, including those that use both surface water and groundwater, is significantly different from the structure of hydropower systems. As a result, water-supply systems require quite different model formulations. Also, the assumptions that are appropriate for managing such systems are significantly different from assumptions that are appropriate for managing hydropower systems.

Hydropower systems frequently are characterized by extensive and highly interconnected networks of reservoirs. Reservoir levels are frequently maintained at or near target levels to provide an optimum hydraulic head for power generation. At a constant head, the benefit of power generation is approximately a quadratic function of the release. Alternate power supplies (e.g., thermal power plants) are usually tied into the system, and the cost of curtailing power deliveries can be tolerated. These characteristics

permit appropriate simplifying assumptions, and various systems analysis methods have been used to produce useful policies for managing hydropower systems.

In contrast, water-supply reservoirs frequently are managed independently or are managed in conjunction with a limited number of other reservoirs. Reservoir levels can vary widely, and it is rarely possible to identify a target storage level. Also, the management goals of water-supply are different and result in different types of functions for measuring benefits. Water supply benefits must be represented by more complex equations that reflect the vital role that water plays in sustaining life. Because of this vital role, extreme events that lead to water shortage have a disproportionate impact on system goals, and, therefore, the value functions are not reducible to simple linear or quadratic equations such as those frequently used in hydropower management. These characteristics make the simplifying assumptions that are appropriate for hydropower systems inappropriate for water-supply systems, and it has been possible to produce useful policies only when managing the simplest systems.

Summary of Optimization Methods

With stochastic inputs, the best management decisions for a system are not obvious. In particular, when future reservoir inflows are variable and uncertain, we cannot easily identify the best water supply and flood release decisions. Also, we cannot correctly evaluate the benefits of system expansion or other changes without stating what the control policies for the modified system are. Since we don't have the benefit of hindsight in making decisions about future events, the best decisions we can make are those that achieve the best expected performance over all possible future scenarios (as discussed in Chapter Two) weighted by their likelihood of happening.

Complex real-world systems are often impossible or very difficult to model and solve analytically. To find a management solution, it is often necessary to excessively simplify the model of a real system. The so called "optimum" solution of this oversimplified model has frequently been far from the true optimal solutions of the real system. According to *Rogers and Fiering* [1986], practical applications of optimization to reservoir management have been limited, and managers have continued to rely largely on experience, in spite of several decades of effort trying to develop systems analysis methods. Indeed, experience appears to have been very effective in identifying appropriate ways to manage many well-established systems [*Bredehoeft et al.*, 1995; *Kelman et al.*, 1990] whose system dynamics, structures, and goals don't change much.

However, this experience is not a sufficient to guide for the management of systems whose dynamics, structure, or goals change. These require systems analysis

methods to determine appropriate real-time controls and correctly evaluate the benefits and costs of different planning options. In particular, we need controls that hedge by balancing short-term costs with long-term expected costs that include the potential costs of extreme events.

This chapter will touch upon some of the optimization methods that can be applied to a stochastic dynamic control problem. The methods vary depending on the number of state variables used to model the system and the complexity of constraints, objectives, and other characteristics. As more detail is added to the model, we may often be forced to accept simplifying assumptions to the point that the optimal solution of the model is useless as a management policy for the real system. The name of the game is to choose a model with enough detail as to be representative of the real system and to choose a solution method that can find the optimal solution.

Table 3A1 summarizes many of the optimization methods that have been applied to reservoir management. These are methods that require (1) a forecast of stochastic inputs, (2) prior knowledge of appropriate control policies, or (3) a function that estimates the expected cost of future operations. Each of these methods can be modified to allow its application to almost any system, but the resulting control-policy will not be useful if the assumptions of the method are not valid. Table 3A1 also identifies for each method the maximum number of state variables that we may use to model a system.

Not included in Table 3A1 are various aggregation-decomposition methods or hierarchical methods that may be used to simplify the model of a system and to allow application of systems analysis to large-scale systems. Aggregation-decomposition methods are those that "aggregate" information used to describe a system's state before optimization and that "decompose" this information following optimization. For example, this may be accomplished by optimization of "subproblems" used to feed to a global optimization problem or by identification of "principal components" of the state information [Saad and Turgeon, 1988]. Lamond and Boukhtouta [1995] summarize some of the aggregation-decomposition methods that may be employed. Hierarchical methods [Pereira and Pinto, 1983] are those that may be used to develop policies for long stages (e.g., weekly or monthly) which are consistent with policies appropriate for short stages (e.g., hourly or daily).

Table 3A1. Optimization Methods Used for Stochastic Dynamic Control

Method	Maximum number of states and other characteristics
<i>Forecast Based Methods</i>	
Accurate if quadratic objective, linear dynamics, normal distributions, and no inequality constraints	
Deterministic Feedback Control	1000
First-Order Analysis	1000 - allows use of third-order polynomial objective
Chance Constraints	1000 - allows use of probabilistic constraints
Linear-Quadratic Control	1000
<i>Parametric Methods</i>	
Accurate if prior knowledge of control policy, quadratic objective, linear dynamics, normal distributions, and no inequality constraints	
Regression	1000 - requires prior knowledge of control policy
Neural Network	100 - requires prior knowledge of control policy
<i>Stochastic Dynamic Programming Methods</i>	
Accurate for any type of objective, system dynamics, inequality constraints, or probability distributions	
Parametric DP	100 - requires prior knowledge of cost-to-go
Discrete DP	10 - requires sufficient state discretization
Stochastic Dual DP	100 - requires sufficient number of trial states, strictly convex or concave cost-to-go
Static	1000 - requires steady-state

B. FORECAST-BASED METHODS

Systems analysis should use optimization methods that incorporate information of the state of the system as the system moves into the future. This allows development of a "feedback" process that uses previous results of stochastic inputs and their impact on the system's state to reduce future uncertainty and improve control decisions. For example, systems analysis applied to reservoir management should use observed inflows and storage levels to reduce the uncertainty of future inflows and improve release decisions.

Forecast-based methods incorporate this feedback using the "best" available estimates of stochastic inputs and states to make control decisions. As time passes, forecasts and control decisions are improved as more and more actual outcomes are observed. Usually, a forecast is "expected" (i.e., the probability weighted) outcome of all possible stochastic-input scenarios. Sometimes the forecast is defined as some scenario that suits the goal of system management. For example, a "worst case" historical scenario of reservoir inflows could be used to make control decisions that will avoid some undesirable outcome (e.g., complete emptying or filling of a reservoir) given a repeat of any previously observed inflows.

Deterministic Feedback Control (DFC)

DFC is the most popular forecast-based method. Control decisions are found by optimizing a deterministic model using expected forecasts and by periodically updating the forecasts and reoptimizing.

Because DFC can be used to generate control decisions for every state, we could develop policies that describe control decisions as a function of state variables. With such a set of policies at hand, decisions could be updated automatically to achieve "closed loop" control. This is rarely done in practice as this requires, in advance, the solution of an optimization problem for each initial and future state of the system. As a result, the updating of control decisions using DFC is not automatic, and forecast-based methods produce "open loop" control [Lamond and Boukhtouta, 1995, p 16]. We must periodically reoptimize as we observe the actual outcome of inputs and states.

DFC identifies control decisions that assume perfect foresight of future inputs. As a result, control decisions may not be cautious enough in avoiding certain low-probability events that have catastrophic consequences. The forecasts do not warn of the possibility of events happening until after system conditions have evolved to the point that it may

not be possible to avoid the consequences [Kitanidis and Andricevic, 1989]. For example, DFC will delay taking action to avoid the emptying of a reservoir until the forecasts of low inflows and reservoir levels indicate that complete draining is likely.

Even though managers of many reservoir systems use some form of DFC, they also recognize that it is important to be extra cautious. Thus, managers often use ad-hoc procedures to hedge against unacceptable, low-probability outcomes [Kitanidis, 1987]. Often, these procedures constrain system operations to avoid the worst consequences under a simulated recurrence of extreme historical events. For example, water-supply agencies in western North America often use the severe 1976-77 drought to evaluate their system's vulnerability [EBMUB, 1992].

While incorporating additional caution, ad-hoc procedures may leave systems vulnerable to extreme events beyond those previously seen. For example, before 1976, the prolonged droughts of the 1920's and 30's were the worst drought periods on record [Glantz, 1982], and the short, but more severe, 1976-77 drought caught many water management agencies unprepared. Preparation for specific extreme historical events—whose exact duplication is nearly impossible—may produce controls that are not cautious enough, too cautious, or both depending on the conditions.

First-Order Analysis Methods

First-order analysis (FOA) methods [Stengel, 1994, pp. 436-443] can be used in place of ad-hoc procedures to induce extra caution and protect against low-probability events. FOA identifies cautious control decisions by including an extra term to incorporate the first-order impact of stochastic inputs. We may expand our representation of a control policy [Kitanidis, 1987] by the series

$$\mathbf{U}_{(t)}^* = \mathbf{U}_{(t)}^{(0)} + \mathbf{U}_{(t)}^{(1)} \sigma + \mathbf{U}_{(t)}^{(2)} \sigma^2 + \dots$$

where $\mathbf{U}_{(t)}^{(0)}$ is the deterministic control policy identified by DFC. Higher-order terms are computed from a Taylor series expansion, assuming no constraints are binding (constraints are applied only to the deterministic policy). It turns out that $\mathbf{U}_{(t)}^{(1)}$ is zero, so FOA uses the deterministic and second-order terms. In contrast, DFC makes the assumption that the higher-order terms are insignificant and that mean values of future inputs are sufficient to describe future uncertainty.

Higher order terms result from the impact of stochastic inputs where σ is a scaling factor of the forecasting-error covariance matrix

$$\mathbf{Q}_{(t)} = E_{\mathbf{w}_{(t)}} \{ \mathbf{w}_{(t)} \mathbf{w}_{(t)}^T \}$$

and

$$\sigma^2 = \text{Trace}\{ \mathbf{Q}_{(t)} \}$$

As with DFC, control decisions depend mainly on the forecast of future states and stochastic inputs. When σ^2 is zero (or when the condition of "certainty equivalence" holds), DFC and FOA methods are exact and produce the same control policy.

By including higher order terms, we can solve a control policy that is better at incorporating the impact of low-probability events and therefore is more cautious. *Kitanidis* [1987] introduced FOA, also known as the "small-perturbation approximation," for the solution of stochastic reservoir control problems. FOA accounts for contingencies and, as a result, is more cautious. *Kitanidis and Andricevic* [1989] showed that policies obtained from FOA perform much better than policies obtained from DFC. Even though DFC generates control decisions that usually perform slightly better than FOA under average conditions, they could perform much worse under extreme conditions.

Chance Constraints

Chance constraints require that operations "succeed" with a specified level of reliability by approximating the probability distribution of stochastic inputs using first and second moments (i.e., mean and variance). Chance constraints do not consider the impact of extreme events when minimizing the expected cost of operations; however, they develop operating strategies that are more cautious than "risk-neutral" strategies that result from using expected values of stochastic inputs. For example, *Revelle et al.* [1969] and *Bhaskar and Whitlatch* [1987] applied chance constraints in their evaluations of linear release-policies constrained by reservoir capacity. Even when feedback is not important, chance constraints are useful in the presence of uncertain parameters. For example, *Wagner and Gorelick* [1987] require that groundwater quality standards be met with a specified level of reliability in the presence of uncertain aquifer properties.

Chance constraints are especially useful in problems where the goal is to minimize the rate of system "failure," defined perhaps as an inability to meet all demands or avoid flood damage. However, water-supply agencies are finding it impossible to avoid "failure" as water rationing or other forms of failure become more common with increasing demands and increasing regulatory constraints on operation. Increasingly, efficient management of reservoir systems is shifting from reducing the likelihood of failure to reducing the severity of failure. Though both FOA and chance constraints use a "first-order" approximation of probability distributions, chance constraints evaluate only the probability of achieving some goal or violating some constraint.

Linear-Quadratic Control

When a single nominal (i.e., optimal) target state can be identified, stochastic inputs tend to move the system away from this state. In this case, optimal control acts to move the system back to the nominal state [Stengel, 1994, pp. 443-451], and we can use linear-quadratic control (LQC) to identify real-time decisions. In unconstrained systems with a quadratic objective, linear dynamics, and normal distributions, the optimal control policy is a linear function of deviations from the nominal state. Such systems allow us to determine optimal control policies without the need to reoptimize after each updated forecast. Thus, we can achieve closed-loop control.

LQC is applicable to a wide variety of problems where the purpose of control is to correct small deviations from a pre-specified desired trajectory or condition. In such cases, small deviations allow us to ignore constraints under most conditions and use a quadratic first-order function to estimate the cost of deviations. For example, *Wasimi and Kitanidis* [1983] and *Loaiciga and Marino* [1985] use LQC for daily operation of a system of flood-control reservoirs to maintain reserved space, *McLaughlin and Velasco* [1990] use LQC to track power output targets in a system of hydropower reservoirs, and *Georgakakos* [1989a] uses LQC in a multiobjective system.

Advantage of Forecast-Based Methods

Forecast-based methods, and particularly DFC, are popular because they are relatively simple to understand and implement. Because these methods use a single scenario of future inputs, only a single optimal control schedule needs to be evaluated and not entire policies. Also, it is much easier to find the optimal control decisions that require evaluation over a single outcome (a "deterministic" problem) rather than control decisions that require evaluations over the entire space of possible outcomes (a "stochastic" problem).

Forecast-based methods can be solved using a variety of deterministic optimization programs. Many of these methods are capable of modeling systems with many state variables and non-linear objective functions, dynamics, and constraints. Among these methods, differential dynamic programming is perhaps the most flexible; and it has been applied to a wide variety of forecast-based optimization problems, including hydropower generation, air traffic control, and various other feedback control problems [Yakowitz, 1982]. Differential DP has also found frequent application in multireservoir control and groundwater flow problems characterized by complex dynamics and abundant state information. For example, *Murray and Yakowitz* [1979]

use differential DP to solve a ten-reservoir control problem, and *Jones et al.* [1987] use differential DP to control nonlinear groundwater hydraulics.

Chang et al. [1992] and *Whiffen and Shoemaker* [1993] modify the DFC method to incorporate feedback laws that change pumping rates directly in response to observations. When systems have linear dynamics, we may also use linear programming or linear-quadratic programming to allow even more large-scale (i.e., detailed) models or to reduce the computational burden. For example, *Atwood and Gorelick* [1985] linearize the response of hydraulic head to determine pumping and recharge schedules for gradient control using a combination of simulation and linear programming.

When caution is important, FOA (or chance-constraint) methods can be used to add caution to DFC solutions. FOA is somewhat more difficult to implement than DFC because we must adjust the deterministic control policy by a hedging term. The computational effort is not significantly greater, and the method can also be applied to problems with many state and control variables. As with DFC, FOA has found application in multireservoir control and groundwater flow problems, especially those characterized by complex dynamics and a large amount of data required to describe the state of a system. For example, *Kitanidis and Andricevic* [1989] solve a four-reservoir problem in which inflows are uncertain. *Andricevic and Kitanidis* [1990] solve a one-dimensional flow problem with uncertain parameters to remove groundwater contaminants. *Lee and Kitanidis* [1991] solve a more detailed flow problem with a longer time horizon. *Georgakakos and Vlatas* [1991] use the FOA method to manage two confined aquifers using various performance indices in the presence of uncertain transmissivity and boundary conditions.

Simplifying Assumptions of Forecast-Based Methods

Forecast-based methods assume that optimization of a deterministic model using a single forecast will yield reasonable control decisions. This assumption is valid only when controls need not incorporate extra caution or "hedging" to avoid the consequences of extreme events; that is to say, when the condition of "certainty equivalence" holds [*Kitanidis and Andricevic*, 1989].

Certainty equivalence holds for unconstrained system models with linear dynamics, quadratic cost functions, and normally distributed inputs [*Kitanidis*, 1983; *Lamond and Boukhouta*, 1995, p 17]. In these cases, we can achieve close-loop control by identifying a control policy by LQC. This can be seen in the results of *Bhaskar and Whitlatch* [1980]; they observe that linear control policies result when using a two-sided quadratic loss function but not when using a one-sided function. Certainty equivalence

also holds approximately for system models whose stochastic inputs cause only small deviations from a nominal state, even when other conditions are not satisfied.

When the condition of certainty equivalence does not hold, FOA (or chance constraints) can be used to add caution to a deterministic control policy. FOA is accurate when σ^2 is "small" in some sense and when fourth and higher derivatives of the value function have little effect on the control policy [Kitanidis, 1987]. As with DFC, FOA "is appropriate for real-time operation problems for which the optimal release depends mainly on the best projection of future conditions (mean values) and the mean squared estimation error of forecasting uncertainty (variances/covariances)." In other words, FOA is appropriate for problems with loose constraints and objective functions that are locally third-order functions. For other problems, we are not guaranteed that control decisions will have the appropriate level of caution.

In Chapter Nine, we will see that DFC can produce reasonable solutions when the condition of certainty equivalence holds approximately, even when a system model contains constraints. Chapter Nine provides a test of DFC solutions obtained by generating a large number of random scenarios. Application of DFC to these scenarios shows that when the condition of certainty equivalence holds at least approximately, the observed performance of DFC is distributed about the performance anticipated by the forecast. Nevertheless, we will also see that DFC solutions deteriorate as the impact of constraints and cost functions moves a system model away from the condition of certainty equivalence.

Application of Forecast-Based Methods to Water-Supply Management

Most models that deal with reservoir system operations assume that the streamflow forecast is error-free [Kelman *et al.*, 1990], permitting identification of a single set of "optimal" control decisions. This allows application of a variety of successive approximation methods to solve high-dimension reservoir control problems that are not tractable using other methods [Foufoula Georgiou, 1991]. In particular, managers of reservoir systems often use some form of DFC because of its ease and flexibility. For these systems, identification of the best forecasts of future inputs is a major priority [Bender and Simonovic, 1994; Georgakakos, 1989b]. In systems that allow identification of a nominal state (such as systems that have a target reservoir or river level for short-term regulation or for hydropower generation, recreation, and navigation), LQC can frequently be applied.

Unfortunately, forecast-based methods are inappropriate for many water-supply management problems and for a variety of other reservoir control problems. In these

problems, we cannot identify a nominal state. For example, we cannot identify a nominal reservoir level for water supply unless reservoir capacities are large relative to demands and inflow variability. Also, the condition of certainty equivalence is not sufficiently correct because extreme events can produce extreme, non-quadratic costs: complete draining of a sole-source reservoir in a water-supply system is catastrophic for consumers of that supply. As a result, water-supply management requires cautious control policies that hedge against severe shortages by imposing rationing early.

Just as it is common to buy insurance that compensates us for the consequences of fire, theft, accidents, or other unlikely events that could have severe consequences in our personal lives, so it is for water-supply management; we buy “insurance” through decisions that hedge against the severe consequences of extreme droughts, floods, or other consequences. Nevertheless, we may not always be rewarded for our caution in managing reservoir systems [Glantz, 1982], and we should balance the cost of hedging with its long-term expected benefit.

As noted earlier, DFC solutions can significantly underestimate the expected cost of system operations because forecasts fail to adequately consider the impact of extreme events. When the impact of low-probability events is significant, DFC policies perform poorly when compared with policies obtained from other more-cautious methods. These other methods hedge against the occurrence of various contingencies [Kitanidis and Andricevic, 1989]. FOA is one example of these other methods; however, FOA captures only the first-order effects of uncertainty, and the resulting decisions may not be cautious enough or sometimes even too cautious.

C. PARAMETRIC METHODS

Parametric methods present another group of optimization methods that we may use to identify feedback control policies. These methods offer some of the advantage of FOA by identifying decisions that are more cautious than those of DFC, and these methods offer some of the advantage of LQC by identifying control policies that allow closed-loop control.

“Parametric” methods reduce the problem of stochastic control to one of fitting a limited number of parameters (e.g., coefficients) in pre-determined control-policy functions. Similar to forecast-based methods, parametric methods incorporate feedback by adjusting the control response to observed outcomes of stochastic inputs and state variables. Unlike previous methods however, parametric methods do not require a forecast and control decisions are directly given as functions of the system state alone.

Parametric methods dramatically reduce the effort required to solve a stochastic control problem by using “soft” knowledge. For example, if we have some expectation based on prior experience about the characteristics of a good control policy function, then we may a-priori identify it as the correct functional form. Another example is modeling which involves fitting parameters to underlying functional forms that identify a system’s structure and operation. As a result, parametric methods are frequently used in combination with other stochastic methods to simplify system models.

To identify a control function, parametric methods identify parameters that match some known or assumed relationship between inputs (the system’s state) and desired outputs (the “optimal” control decisions). In some cases, this relationship can be captured by arbitrary functions that have no basis in the structure or operation of a system, since the application of these functions is judged solely by their ability to empirically reproduce the relationship between inputs and desired outputs. These outputs are usually identified by deterministic optimization of historical inputs (e.g., reservoir inflows) or other scenarios. A stochastic problem is solved when optimization identifies parameters that minimize some measure of “cost,” such as those that minimize the sum of squared differences between control policy outputs and desired outputs.

Following is a brief outline of stochastic optimization methods that are “parametric” because the solution process includes “fitting” parameters to pre-selected functions. The process of fitting parameters to pre-defined control-policy functions gives rise to methods that share similar advantages and limitations.

Regression

Regression includes a wide variety of curve fitting methods. The particular function form of the curve and the particular fitting method depends on what prior knowledge we have about a control policy solution. Regardless of the form and method employed, the basic approach is to reproduce the relationship between sets of input and output data. The simplest example is linear regression that describes a relationship between a single input x and output y by fitting parameters $\{a,b\}$ in the equation

$$\hat{y}(x) = ax + b$$

to data pairs $\{(x^{(i)}, y^{(i)}), i=1, N\}$.

Usually, a smooth curve is unable to exactly match input and output data. For example, random errors (that add “noise” to the relationship) and imprecise knowledge about the correct functional form prevent a complete description of the relationship. To identify parameters, we minimize some measure of the errors between true values $y^{(i)}$ and

estimates $\hat{y}^{(i)}$. Typically, the best fit is identified by linear-quadratic programming to minimize the sum of squared errors. When a better fit is required, we might use higher order functions or functions that seem more appropriate based on the observed or the expected relationship between x and y .

Neural Networks

Neural networks also describe a functional relationship between sets of input and output data, but they require somewhat less prior knowledge to identify a control policy. A neural network consists of layers of nodes or “neurons” that sum input signals to produce an output signal. The “input” layer is the first layer of nodes that accepts signals from the system’s state. The “output” layer is the last layer of nodes that produces control decisions. In between, there may be one or more “hidden” layers that combine these signals. The more hidden layers and nodes used to model a system, the more degrees of freedom that a control policy has to take into account in order to capture the interrelationship between inputs and outputs. Input and output data are used to “train” the neural network in an iterative process that varies the weights applied to signals between nodes. The objective is to find weights that minimize some measure of the errors between output from the network and the desired output from the data sets. When a better fit is required, an improved neural network model is developed with more nodes in a layer, more layers, or different methods of summing input signals.

Advantage of Parametric Methods

As with forecast-based methods, parametric methods are relatively simple to implement and understand. By using soft knowledge about a control policy solution, parametric methods allow us to significantly reduce the effort required to solve a stochastic control problem. Also, parameters can be fitted by a wide variety of deterministic optimization programs, and parametric methods can be applied to large-scale stochastic control problems.

In some applications, parametric methods avoid the development of stochastic models. This can be useful in cases where it is difficult to construct a stochastic model. For example, the historical record may be insufficient to build a detailed stochastic model for inputs that are highly correlated and seasonal. If the historical record is sufficiently long, there is no need to develop synthetic scenarios to represent the relationship between state-variable inputs and control-variable outputs.

Simplifying Assumptions of Parametric Methods

There is no guarantee that solutions obtained by parametric methods are truly optimal because the control-function shape specified before design may be incorrect [Stengel, 1994, p. 185]. Also, there are no general guidelines for function selection, other than to try to use functions that are easy to implement numerically or that are "natural" for the problem at hand [Stengel, 1994, p. 193].

Unfortunately, it may be difficult to test whether a particular functional form is appropriate since we may only compare solutions using different functions (based on the same soft knowledge). Unlike DDP (described later) which allows an increasingly fine mesh of discrete states, there may not be a series of increasingly detailed functions that can be used to generate solutions that converge to the true optimal solution. In addition, increasingly detailed functions can lead to the phenomenon of overfitting as the number of degrees of freedom in the solution structure approaches the number of available pairs of input and output data.

Because the optimal relationship between the control policy and inputs is unknown (since this is what we want to determine), it is common practice to use the historical record in a deterministic optimization and to assume that the resulting relationship can be used to approximate the true optimal relationship [Karamouz and Houck, 1982; Karamouz and Houck, 1987; Young, 1967]. This approach is most popular both because it can provide the needed data sets and because it avoids the need for stochastic models of inputs. However, where the historical record is short or is not representative of future conditions, synthetic scenarios can be generated [Karamouz et al., 1992].

Unfortunately, the relationship between input and output data established by deterministic optimization is not correct unless the condition of certainty equivalence holds. Because deterministic optimization uses perfect foresight, the true optimal relationship is the same as the average relationship only when the condition of certainty equivalence holds. When a system is far from satisfying this condition, policies that result from parametric methods may generate an incorrect relationship.

Application of Parametric methods to Water-Supply Management

Parametric methods have found considerable application to reservoir management problems in the literature. A significant advantage of these methods is that they can be applied to systems with well-established simulation models or in combination with other optimization methods to solve relatively large-scale reservoir management problems.

Parametric methods often permit significant simplification by using soft knowledge that is often available from long experience operating these systems.

However, without soft knowledge that can be used to identify the functional form of a control-policy function, parametric methods should not be used. Parametric methods used in systems analysis of new and unfamiliar problems can result in sub-optimal control policies. Moreover, it is likely that a fitted control-policy function will perform poorly and result in significant losses under critical extreme conditions where intuition is weak.

In addition, parametric methods should be applied cautiously to avoid over-fitting limited data. This is especially true when we use detailed models, such as when we use high-order control-policy functions, or when we use numerous layers and nodes in neural networks. The worst models are often the most complex, making it difficult to interpret solutions. Complex models also make it difficult to verify that control policies are consistent and otherwise in accord with our understanding of how the system should be operated. On the other hand, the best models are often the simplest. This is particularly true when data values are not known with much accuracy (e.g., due to measurement error) and detailed models may result in fitting the noise rather than the true relationship. Careful model selection can often simplify the representation of a system without significantly degrading the accuracy of solutions. For example, when a system contains seasonality or other non-stationary process (e.g., a process that changes, such as from changing system structure or changing probability distributions), it may be appropriate to “pre-whiten” the data, removing the influence of trends or other changes.

In the case of the real-time control of reservoir systems, the input data are the reservoir storage, prior inflows, demand, and other state information, and the outputs are the amount to release and other control decisions. The most popular regression method applied to the control of reservoir systems is multiple linear regression that describes the relationship between the vector of inputs \mathbf{x} and a control decision $u = U(\mathbf{x})$ given by

$$U(\mathbf{x}) = a_0 + \sum_{j=1}^n a_j x_j$$

where $\{a_j, j=1, n\}$ are fitted parameters, or weights, applied to each state variable. *Young* [1967] introduced the use of linear regression to determine release decisions that are a linear function of inflow and storage levels. *Young* [1967] and *Bhaskar and Whitlatch* [1980] tested more complex functions and found that linear functions often perform better in many cases. Other applications have added chance constraints and more complex models [*Bhaskar and Whitlatch*, 1987; *Curry et al.*, 1973; *Karamouz and*

Houck, 1982; Loucks and Dorfman, 1975; Marino and Simonovic, 1981; Revelle et al., 1969; Simonovic, 1979], though most applications continue to use linear control-policy functions. Bogle and O'Sullivan [1979] take a somewhat different approach by identifying parameters of a pre-determined class of control-policy functions. They constrain the control policy for reservoir releases to be a step function with two parameters that identify critical storage levels below which releases are at a minimum value and above which releases are at a maximum value, with linearly changing releases between these storage levels.

Neural networks have recently received considerable attention for the real-time control of reservoir systems [Saad et al., 1994] and for the development of forecasts [French et al., 1992; Karunanithi et al., 1994; Tang and Fishwick, 1993]. Raman and Chandramouli [1996] provide a clear and concise description of the method's application to a simple reservoir problem. In addition, neural networks find application to a variety of large-scale problems characterized by a large amount of state information. These applications include aquifer parameter estimation [Rashid et al., 1992; Rizzo and Dougherty, 1994], groundwater remediation [Ranjithan et al., 1993; Rogers and Dowla, 1994], and aggregation/decomposition of large-scale reservoir systems [Saad et al., 1996; Saad et al., 1994].

D. STOCHASTIC DYNAMIC PROGRAMMING METHODS

Stochastic dynamic programming (SDP) presents a third group of optimization methods that we may use to identify feedback control policies. Bellman [1957] coined the term "dynamic programming" (DP) to describe "the theory of multistage decision processes." Yakowitz [1982] noted that DP may broadly define all optimization methods that solve time-varying or dynamic problems; however, conventional usage of the term identifies optimization methods that solve a dynamic problem by dividing it into a series of subproblems, one for each stage of an operating horizon.

In this section, we will consider SDP methods that develop explicit cost-to-go functions for each stage of a stochastic control problem. If we consider enough discrete states, then we can identify functions that accurately describe the expected cost of future operations for any initial state. By estimating a cost-to-go function for each stage of a stochastic control problem, SDP methods can decompose the difficult problem of finding optimal control policies for equation (2C12) into a series of easier subproblems, one for each stage. In contrast, deterministic DP methods (such as differential DP) do not develop explicit cost-to-go functions since they need only identify a single control

schedule and not a series of control policies. Note that SDP methods can also be applied to deterministic problems, though it is not usually practical to do so.

In each stage of the problem, the control policy is defined by decisions that minimize the current cost of operations and the expected future cost given by the cost-to-go function for the next stage. This means that SDP problems are usually solved using backwards recursion, starting with the last stage (as the first subproblem) and working backwards to the first stage. At the beginning of a stage, the expected cost for each state is evaluated as the minimum weighted average using various scenarios to go from the beginning state to a state in the next stage. The effort to calculate this expected cost and to identify optimal control decisions is much less than the effort to calculate the expected cost and control decisions for a scenario path that spans all stages of a control problem. A more detailed illustration and discussion of SDP follow in the next chapter.

The principal challenge in applying SDP is estimating a continuous cost-to-go function from discrete costs. SDP methods are distinguished by how they interpolate between discrete costs and by the number of initial states that must be evaluated. Following is a brief outline of SDP methods that use different function estimates to solve stochastic control problems. Unlike forecast-based methods and parametric methods discussed earlier, these methods have widely varying abilities and limitations.

Parametric Dynamic Programming

Parametric DP uses soft knowledge of the cost-to-go to identify pre-defined forms of a cost-to-go function. Thus, the method is similar to parametric methods because it requires some prior knowledge of a solution and because it requires fitting parameters. However, control decisions are the direct result of optimizing system goals and not the indirect result of a control policy fitted to reproduce a relationship between input and output data sets. Because parametric DP does not require input and output data sets, accurate solutions do not require the condition of certainty equivalence (so that the data sets identify the optimal relationship) and they avoid the problem of overfitting.

As a result of using soft knowledge, parametric DP dramatically reduces the effort required to solve stochastic control problems. Similar to parametric methods, parametric DP requires evaluation of only a limited number of discrete states to fit parameters (if we have identified the correct functional form and if there is no noise). In general, the number of discrete states grows only linearly with the number of state variables. For example, $(n+1)$ initial states are required to fit an n -dimensional multi-linear cost-to-go function.

Bellman and Dreyfus [1962] were the first to propose parametric DP to overcome the so called curse of dimensionality. When solving a stochastic control problem, the curse of dimensionality refers to the exponential growth in effort with the number of state variables. *Bellman and Dreyfus* suggested using orthogonal polynomials to provide a global approximation of the cost function. However, their purpose was to reduce the exponential growth in memory required to store cost data at grid nodes, and not specifically to interpolate a cost-to-go function accurately. Unless polynomials are appropriate for global approximation, cost-to-go estimates are not accurate and the use high-order polynomials to match data may produce non-convex cost-to-go functions that oscillate. Other authors have recognized that local approximations provide better cost-function estimates [*Foufoula Georgiou*, 1991].

Gal [1979] applied parametric dynamic programming to a water supply system with three state variables (one for storage and two for inflows during two prior stages) by developing an iteration method to identify parameters of second-order polynomial cost-to-go functions. However, *Gal* noted that “contrary to the usual dynamic programming approach, the parameter iteration method is not fully automatic. It is expected that the user have a good understanding and intuition about the behavior of the considered system.” Nevertheless, we may apply parametric DP to problems for which the curse of dimensionality prevents application of DDP.

Discrete Dynamic Programming

Discrete dynamic programming (DDP) uses interpolation over a fine mesh of discrete states to identify a cost-to-go function. In contrast to parametric DP that finds a single global function that is applied over the entire state domain, DDP uses a local approximation. Early applications of DDP depicted the cost-to-go as a set of nodes in a directed-flow network, as in the traditional “stagecoach problem” [*Hillier and Lieberman*, 1990]. Accurate solution development required a large number of discrete states at nodes of a fine state-space grid, and continuous cost-to-go functions were estimated using “nearest neighbor” interpolation (using a value at the nearest discrete state to estimate the costs at intermediate states). In more recent applications, multi-dimensional linear interpolation or higher-order interpolation methods [*Foufoula Georgiou and Kitanidis*, 1988; *Johnson et al.*, 1993; *Kitanidis and Foufoula Georgiou*, 1987] have been used to estimate the costs at intermediate states.

Higher-order interpolation methods can be more accurate and may produce accurate cost-to-go estimates with coarser state-space grids. *Kitanidis and Foufoula-Georgiou* [1987] and *Foufoula-Georgiou and Kitanidis* [1988] developed Gradient DP to

use both cost and gradient information to improve interpolation over a rectangular grid. Gradient DP uses cubic Hermite polynomials in an interpolation algorithm that preserves costs and gradients at nodes of the grid. They demonstrate that with a decrease in the grid discretization interval of Δx , the error of the control policy and the cost functions converge respectively as $(\Delta x)^3$ and $(\Delta x)^4$ using Hermite interpolation versus Δx and $(\Delta x)^2$ using linear interpolation. Their implementation of Hermite interpolation produced a cost-to-go function estimate that was continuous but only piecewise smooth. *Johnson et al.* [1988; 1993] avoid the need for gradients by developing cubic spline interpolations that use node values distributed over a larger subdomain. The resulting interpolation algorithm produces continuous first and second derivatives and allows them to fully implement an efficient Newton-based search method.

Stochastic Dual Dynamic Programming

Stochastic Dual Dynamic Programming (SDDP) estimates a cost-to-go function by evaluating expected costs only where needed most. At each discrete state, an estimated cost and gradient is evaluated to define a “cutting plane.” A cutting plane is an affine function that bounds the optimal cost-to-go function. For a convex cost-to-go function (and minimum-cost objective), a cutting plane is a lower bound. By increasing the number of “cuts,” the cost-to-go function can be estimated with increasing accuracy. Using a linear or linearized model in a single-stage subproblem, cuts can be efficiently identified by the dual to the linear programming problem [*Gorenstin et al.*, 1992; *Pereira and Pinto*, 1991].

SDDP iteratively improves estimates of the cost-to-go for each stage by backward and forward loops using the Benders decomposition principle [*Benders*, 1962]. During a backward loop through the stages, SDDP improves the accuracy of estimated cost-to-go functions by adding cuts. The backwards recursion permits cost-to-go estimates using the improved cost-to-go functions of later stages. During a forward loop through the stages, SDDP uses trial scenarios to evaluate the simulated performance of the current cost-to-go functions and to identify new discrete states for cuts in the next backwards loop. The average cost of operations for the trial scenarios identifies a probabilistic upper bound on the optimal cost-to-go function (assuming a convex function), and the certainty of this bound increases with the number of scenarios. With continued looping backwards and forwards, the function estimate and the simulated estimate converge, providing tighter bounds on the true cost-to-go. Estimates are considered to be accurate when the lower and upper bounds are deemed “close enough” to tightly bound the true cost-to-go with a specified probability.

As with neural networks, SDDP has recently received considerable attention for the real-time control of reservoir systems. *Pereira and Pinto* [1985] used the dual solutions and Benders decomposition to solve a four-reservoir case study and a 37 reservoir example with five month-long stages. *Pereira and Pinto* [1991] coupled dual decomposition to the SDDP method to approximately solve release decisions for a 39-reservoir hydropower system. A variety of other applications have also been made to other hydrothermal power generation problems [*Gorenstin et al.*, 1992; *Jacobs et al.*, 1995; *Rotting and Gjelsvik*, 1992]. More recent applications have added “importance sampling” [*Dantzig and Glynn*, 1990; *Infanger*, 1991] to reduce the variance of upper bounds and to improve convergence of lower and upper bounds by improving the sampling of scenarios. Importance sampling identifies new cuts that are more significant (i.e., that trim off more of the region between the previous function estimate and the optimal function) than those identified by “naive” Monte Carlo sampling.

Static Dynamic Programming

The time horizon for management is often sufficiently long that control policies achieve a steady-state solution. In these cases, longer time horizons or different final cost-to-go functions (i.e., $F_{N+1}(x)$) do not change initial control policies. Such invariant control policies turn out to be myopic [*Lamond and Boukhtouta*, 1995], meaning that there is only one optimal control policy for the immediate future. Myopic policies are particularly common when a discount rate is used to diminish the current value of future operations. With a discount rate, both the cost-to-go function and control policies achieve steady-state solutions.

When the optimal control policies are myopic, an infinite-time-horizon control problem can be solved by a single-stage problem [*Sobel*, 1989]. In the case of seasonal models, an infinite-time-horizon control problem can be solved by a problem with one stage for each season. A variety of methods can then be applied to identify the optimal control policy or cost-to-go function. *Sobel* [1975] notes that myopic release policies are often linear.

Advantage of SDP

SDP methods can identify optimal solutions in spite of inequality constraints, non-linear dynamics, complex value functions and probability distributions [*Birge*, 1995]. In contrast, forecast-based and parameter-iteration methods may produce solutions that are far from optimal, especially if conditions are far from certainty equivalent. Even FOA may not identify the optimal control policy, in spite of its ability to incorporate

caution. In contrast, SDP avoids these assumptions and can identify control policies that are appropriately cautious.

SDP methods can identify the true optimal control policy for a model, and can be used to test simplifying assumptions that may allow solution of large-scale models by other methods. In the worst case, when we have little or no prior knowledge of a solution, we can use DDP. With a sufficiently fine grid of the state space, we can identify a control policy with any desired precision, even if the cost-to-go function is not strictly convex (or strictly concave in maximization problems). In cases where we have some prior knowledge of a cost-to-go function and need to solve problems with more state information, we may use parametric DP, SDDP, or static DP to solve a control policy.

Simplifying Assumptions of SDP

The effort required to apply SDP methods, particularly DDP, grows exponentially with the number of state variables used to model a system (the “curse of dimensionality”) unless we have prior knowledge that allows the use of simplifying assumptions. As a result, SDP is difficult or impossible to apply to new problems requiring a large amount of data to describe the state of a system.

Nevertheless, SDP is often applied in spite of this limitation because of its ability to identify appropriately cautious policies when a system contains constraints or complex objectives, dynamics, or inputs. These applications assume that simple “lumped-parameter” models provide a sufficiently accurate representation of real systems. In such systems, the state information is aggregated into a few state variables. For example, a single reservoir level is often used to represent the total storage of systems with numerous reservoirs [Kelman *et al.*, 1990; Terry *et al.*, 1986]. In addition, a variety of aggregation and decomposition methods may be used to preserve some detailed characteristics of large systems while reducing the number of state variables used in optimization [Saad *et al.*, 1996; Saad and Turgeon, 1988; Saad *et al.*, 1992; Turgeon, 1980; Turgeon, 1981; Valdes *et al.*, 1992].

However, sometimes the level of aggregation may be inappropriate. Aggregation is always a compromise used to reconcile the need for realism and the need to solve a problem. Though we may solve highly accurate numerical solutions using simple models, they may be meaningless for management of real systems [Rogers and Fiering, 1986]. To solve a problem having more state variables than can be handled with DDP, additional assumptions may be employed to allow application of parametric DP, SDDP, or static DP. If we have additional information that permits assumptions appropriate for

forecast-based methods or parametric methods, then we can solve control policies for large-scale models without aggregating system characteristics.

Application of SDP to Water-Supply Management

Application of SDP methods to reservoir management has been a topic of considerable interest beginning with the work of *Bellman and Dreyfus* [1962]. Of the SDP methods subsequently developed, DDP has been of particular interest because of its ability to cope with problems of a general non-linear and stochastic character. As a result, DDP is often considered synonymous with SDP when applied to stochastic dynamic control problems.

SDP allows us to incorporate feedback with fewer simplifying assumptions. Feedback is used to incorporate the dynamic nature of physical inputs such as streamflow and demand. Other methods also incorporate feedback but require additional simplifying assumptions. When these assumptions are not entirely appropriate, these other methods produce suboptimal solutions. For example, *Kitanidis and Andricevic* [1989] show that policies obtained from DDP perform better than policies obtained from FOA and much better than policies obtained from DFC. However, they also observe that we cannot apply DDP to complex system models to which we can apply these other methods.

DP methods also use simplifying assumptions, but usually these assumptions are based on soft knowledge we have about a system. For example, when we have prior knowledge of appropriate cost-to-go functional forms, we can use parametric DP. When we have knowledge that simple lumped-parameter models are appropriate, we can use DDP. Fortunately, we may use some of these assumptions even when we apply systems analysis to unfamiliar problems. For example, in Chapter Eleven we consider the real-time control of conjunctive-use systems using a simple lumped-parameter model. For this problem and similar water-management problems, DDP is the most appropriate method since it uses the least amount of prior knowledge and makes the fewest simplifying assumptions.

Almost all systems have parameters whose estimated values are dynamic because they describe a changing state of knowledge. In reservoir management problems, these parameters are future inputs of streamflow and demand. In groundwater management problems, these parameters are inputs from recharge and boundary conditions, but also include estimates of aquifer characteristics. Though groundwater management problems are constrained and have complex dynamics, the large amount of data required to describe the state of the system requires detailed models that are beyond the ability of current SDP methods.

CHAPTER 4.

DYNAMIC PROGRAMMING

Reservoirs exist to regulate stochastic streamflows, water demands, power demands, and other system inputs. Because of limits on the capacity of reservoirs and because of the potentially high cost of operation during extreme events, there is a strong incentive to make cautious control decisions. For example, cautious water-supply decisions hedge by rationing earlier to balance the short-term cost of this rationing with the long-term expected cost of extreme shortages.

Dynamic programming (DP) methods can identify appropriately cautious controls better than other optimization methods, and these methods provide great flexibility in modeling stochastic and non-linear system characteristics. Among stochastic methods, DP has the advantage of decomposing the overall problem into a number of simpler problems that are solved sequentially. However, DP methods may be unable to identify controls with many state variables. Fortunately, we can characterize a fairly large number of reservoir management problems by limited state information contained in a few "lumped" parameters.

In many cases though, the degree of simplification required is still more than desired. A principle goal and result of this dissertation are the development of techniques that reduce the degree of simplification required. A couple of these techniques are developed in Chapters Five and Seven and the benefit of these methods is evaluated in Chapters Six and Eight.

This chapter provides a detailed explanation and illustration of DP and the particular method of discrete dynamic programming (DDP). The purpose of this chapter is to identify DDP as an appropriate optimization method for reservoir-management problems, particularly for water-supply management. Following a detailed discussion of the DP approach to solving stochastic dynamic control problems, we develop the method and notation for DDP used in later chapters of this thesis. Also, we will observe why the strengths and limitations of DDP make it an appropriate method for management of reservoir systems used for water supply.

A. WHAT IS DYNAMIC PROGRAMMING?

The evolution of a system is identified by its changing state during the operating horizon. This operating horizon can be divided into stages used to identify the impact of a single set of control decisions and other inputs as in equation (2C2). Stages are defined so that a system must pass through each stage in sequence and so that a system cannot reach a future stage without passing through all intermediate stages. Stages usually represent increments of time since we cannot get from a current moment to some future moment without passing in sequence through all intermediate moments. For example, if we currently find ourselves in the month of December, we can only go forward to the month of January and cannot go back to the month of November. Also, we cannot get to the month of April without passing through the months of January, February, and March (Figure 2B1).

DP takes advantage of this division to decompose the problem of minimizing equation (2C12) into a series of easier subproblems, one for each stage. When we describe the state of a system by a finite number of variables, then we assume these variables summarize all necessary information about a system's history. This characteristic allows us to identify optimal control decisions for each stage without any consideration of prior stages; all we need to know is a system's current state. DP takes advantage of this structure by recursively solving subproblems, starting with the last stage and working backwards until we arrive at the first or current stage. The solution of each subproblem identifies a set of control policies and a cost-to-go function that identifies the expected cost to go from any initial state.

If the control policy for each stage is optimal, the control policy for the first stage is optimal for the whole problem. We are assured of this by the "Principle of Optimality" [Bellman, 1957] which states that

"An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

This tells us that the control policy developed for each intermediate stage is an optimal policy for the remaining stages.

To see this more clearly, let us consider the last stage of a DP problem. For this last stage, we represent the total cost function as

$$V_{t_N}(\mathbf{x}, \mathbf{u}, \mathbf{w}) = C_{t_N}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{N+1}}(\mathbf{y}) \quad (4A1)$$

As discussed before, the cost-to-go function $F_{t_{N+1}}$ identifies the relative preference for state $\mathbf{x}_{(t_{N+1})}$ at the end of a time horizon used for system management. The transition function

$$\mathbf{y}_{(t_N)} = \mathbf{T}_{(t_N)}(\mathbf{x}, \mathbf{u}, \mathbf{w})$$

identifies the ending state $\mathbf{y}_{(t_N)} = \mathbf{x}_{(t_{N+1})}$. For any discrete state $\mathbf{x}_{(t_N)}^{(i)}$, we identify optimal control decisions $\mathbf{u}_{(t_N)}^*$ that minimize the total expected cost of operations from the current stage through the last stage. When random inputs are present, we cannot identify the actual total cost of operations that we will observe, so we identify the *expected* total cost. Using optimal controls, the cost-to-go from state $\mathbf{x}_{(t_N)}^{(i)}$ in the last stage is

$$F_{t_N}(\mathbf{x}^{(i)}) = \min_{\mathbf{u}} \{ E_{\mathbf{w}} \{ V_{t_N}(\mathbf{x}^{(i)}, \mathbf{u}, \mathbf{w}) \} \}$$

This assumes we have no foresight of random inputs $\mathbf{w}_{(t_N)}$. Without foresight, we identify a single vector of control decisions $\mathbf{u}_{(t_N)}^*$ that is feasible regardless of outcome $\mathbf{w}_{(t_N)}$. Solving $F_{t_N}(\mathbf{x}^{(i)})$ for a sufficient number of discrete states, we can estimate a cost-to-go function $F_{t_N}(\mathbf{x})$ and control policy $\mathbf{U}_{t_N}^*(\mathbf{x})$ for any initial state.

As discussed in Chapter 2, the lack of foresight may result in control decisions that are unrealistic and over constrained. Instead, we identify controls using limited foresight of current inputs $\mathbf{w}_{(t_N)}$ to identify controls. If we assume a current outcome of these inputs $\mathbf{w}_{(t_N)}^{(k)}$, the optimal controls are those that solve

$$V_{t_N}(\mathbf{x}^{(i)}, \mathbf{u}^*, \mathbf{w}^{(k)}) = \min_{\mathbf{u}} \{ V_{t_N}(\mathbf{x}^{(i)}, \mathbf{u}, \mathbf{w}^{(k)}) \}$$

The expected cost-to-go from state $\mathbf{x}_{(t_N)}^{(i)}$ is then an expected cost of all possible outcomes:

$$F_{t_N}(\mathbf{x}^{(i)}) = E_{\mathbf{w}} \{ \min_{\mathbf{u}} \{ V(\mathbf{x}^{(i)}, \mathbf{u}, \mathbf{w}) \} \} \quad (4A2)$$

Using limited foresight, the control policy is also a function of current inputs $\mathbf{w}_{(t_N)}$. In other words, $\mathbf{u}^* = \mathbf{U}_{t_N}^*(\mathbf{x}, \mathbf{w})$.

With the solution of the t_N subproblem, we now solve the next subproblem to identify a cost-to-go function $F_{t_{N-1}}$ and control policy $\mathbf{U}_{t_{N-1}}^*$ for the second-to-last stage. For the second-to-last stage, the total cost function is

$$V_{t_{N-1}}(\mathbf{x}, \mathbf{u}, \mathbf{w}) = C_{t_{N-1}}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_N}(\mathbf{y}) \quad (4A3)$$

Note that the total cost is only a function of the state, control decisions, and random inputs for the current stage. Future control decisions are already established by the control policy $\mathbf{U}_{t_N}^*$, and the expected cost of these future decisions is summarized by the cost-to-go function F_{t_N} . The expected cost-to-go from state $\mathbf{x}_{(t_{N-1})}^{(i)}$ is

$$F_{t_{N-1}}(\mathbf{x}^{(i)}) = E_{\mathbf{w}} \{ \min_{\mathbf{u}} \{ V_{t_{N-1}}(\mathbf{x}^{(i)}, \mathbf{u}, \mathbf{w}) \} \} \quad (4A4)$$

Solving $F_{t_{N-1}}(\mathbf{x}^{(i)})$ for a sufficient number of discrete states, we can again estimate a cost-to-go function $F_{t_{N-1}}(\mathbf{x})$ and control policy $\mathbf{U}_{t_{N-1}}^*(\mathbf{x}, \mathbf{w})$ for any initial state.

This process can be repeated for each prior stage $t_j = t_{N-2}, t_{N-3}, \dots, t_1$, backwards in time until we reach the first (or current) stage. In each stage, we obtain a cost-to-go function $F_{t_j}(\mathbf{x})$ and a control policy $\mathbf{U}_{t_j}^*(\mathbf{x}, \mathbf{w})$ identified by solving

$$V_{t_j} = C_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{j+1}}(\mathbf{y}) \quad (4A5)$$

$$F_{t_j}(\mathbf{x}) = E_{\mathbf{w}}\{ \min_{\mathbf{u}}\{ V_{t_j} \} \} \quad (4A6)$$

The Principle of Optimality assures us that current control decisions $\mathbf{u}_{(t_j)}^*$ based on the cost-to-go function $F_{t_{j+1}}$ are optimal because all subsequent decisions are optimal. The new cost-to-go function F_{t_j} tells us the expected cost of system operations based on application of optimal control policies in the current and each subsequent stage $t \in [t_j, \dots, t_{N+1}]$.

By decomposing dynamic problems into subproblems, DP reduces the effort required to solve multi-stage problems. As we saw at the end of Chapter 2, the number of stochastic-input scenarios and the computational effort grow exponentially with the number of stages if we apply optimization to equation (2C12) naively. By decomposing the optimization of equation (2C12) into subproblems the effort to solve a subproblem for each stage is independent of the number of stages. As a result, the computational effort of DP grows linearly with the number of stages.

However, the above discussion of DP specifies neither what constitutes a "sufficient" number of discrete states nor how we use solutions to approximate continuous cost-to-go and control-policy functions. How we approximate these functions significantly effects the practical application of DP, as chapter 3 briefly outlined.

The remaining sections provide a more detailed illustration of the particular DP method of DDP and an explanation of limits on its practical application. DDP is an appropriate method for analysis of water-supply management problems. As we mentioned in Chapter 2, DDP is the most general DP method because it requires the least amount of prior knowledge. Also, we observed that water-supply management problems can often be solved using simple lumped-parameter models with limited state information.

B. DISCRETE DYNAMIC PROGRAMMING

To pass a sufficiently accurate estimate of the cost-to-go function to the subsequent subproblem, DDP solves equations (4A5) and (4A6) for a sufficient number of discrete initial states, $\mathbf{x}_{(t_i)}^{(i)}$. These discrete states are located at the nodes of an n -dimensional grid that spans possible states of a system. DDP estimates the cost-to-go at intermediate states using interpolation, and the optimal cost-to-go function $F_{t_i}(\mathbf{x})$ is estimated by interpolating functions that are connected piecewise over the n -dimensional domain \mathbf{x} . The optimal control policies $\mathbf{U}_{t_i}^*(\mathbf{x}, \mathbf{w})$ can be similarly estimated or can be directly solved by minimizing equation (4A5). By using finer grids (and increasing the number of discrete states), we can estimate the cost-to-go and control decisions with any desired level of accuracy.

1. Illustration of the Last Stage Subproblem

For each discrete state $\mathbf{x}_{(t_N)}^{(i)}$, we estimate the expected cost-to-go, $F_{t_N}(\mathbf{x}^{(i)})$, by applying equations (4A1) and (4A2). This is accomplished by looping through a sufficient number of outcomes $\mathbf{w}_{(t_N)}^{(k)}$ and finding the optimal decisions $\mathbf{u}_{(t_N)}^*(\mathbf{x}^{(i)}, \mathbf{w}^{(k)})$ for each. The expected cost-to-go is the probability weighted average of the cost for each outcome.

If we solve $F_{t_N}(\mathbf{x}^{(i)})$ for a sufficient number of discrete states, we can estimate the continuous functions $F_{t_N}(\mathbf{x})$ for any state. Figure 3B1 illustrates a grid of discrete storage levels (i.e., states) $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\Lambda)}\}$ that we could use to model the single-reservoir problem of Figure 2B1. Figure 3B1 also illustrates the state evolution from each discrete storage level to a desired zero storage level at $t = t_{N+1}$. In this illustration, control decisions are inflexible since the policy is to release all water, regardless of initial storage level.

If inflows are stochastic and have a distribution such as illustrated in Figure 3B2, we need an appropriate numerical integration method to evaluate $F_{t_N}(\mathbf{x}^{(i)})$ for each discrete state $\mathbf{x}_{(t_N)}^{(i)}$. Even though control decisions are inflexible in this last stage, the costs and constraints that apply to these decisions are stochastic. To solve equation (4A6) using numerical integration, we use

$$F_{t_N}(\mathbf{x}^{(i)}) = \sum_{k=1}^M \nu_k \{ \min_{\mathbf{u}} \{ V_{t_N}(\mathbf{x}^{(i)}, \mathbf{u}, \mathbf{w}^{(k)}) \} \}$$

The weights ν_k and outcomes $\mathbf{w}^{(k)}$ (also known as "abscissas") are provided by the particular numerical integration method that we select. The most common is the

trapezoidal method outlined by equation (2C10) which allows use of any desired grid of abscissa values.

For each node $\mathbf{x}_{(t_N)}^{(i)}$ of the grid, Figure 3B3 illustrates possible values of $F_{t_N}(\mathbf{x}^{(i)})$ that we might evaluate. By interpolating these values at intermediate states, we can produce a continuous estimate of the true function. For this illustration, the best storage level is somewhere in the middle; high costs at extreme low and high storage levels could represent the risks of water shortage and flooding when these are both goals of a system.

Figure 3B1. Discrete States and State Trajectories for the Last-Stage Subproblem

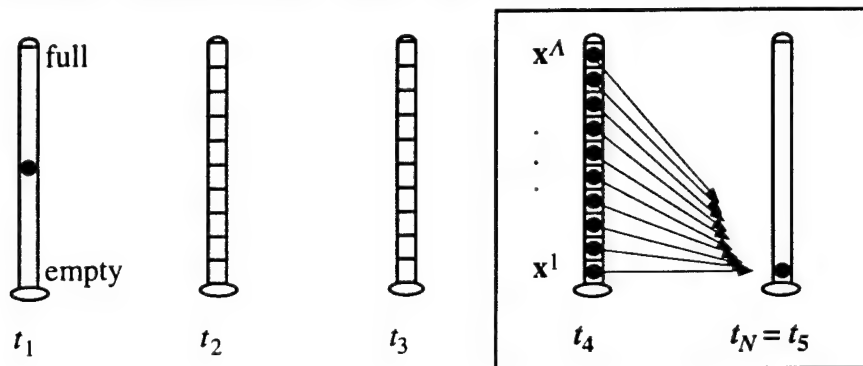


Figure 3B2. Sample Streamflow Distribution

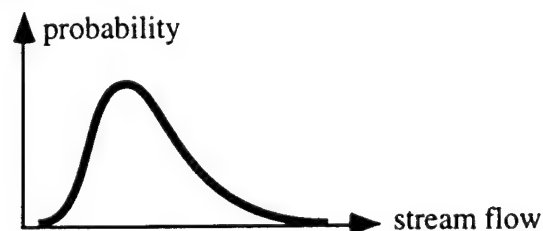
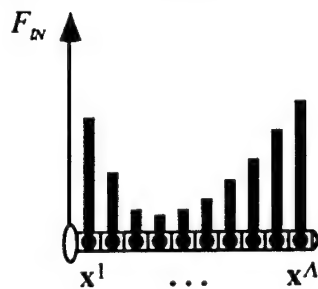


Figure 3B3. Discrete Estimate of the Last-Stage Cost-To-Go Function

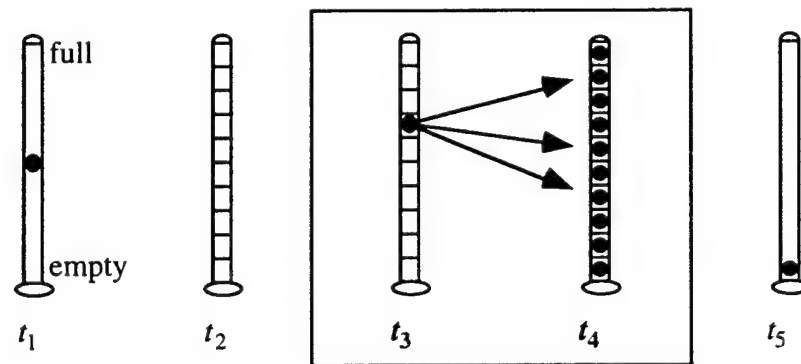


2. Illustration of the Remaining Subproblems Using Recursion

Once we have estimated the last-stage cost-to-go function, we can applying equations (4A3) and (4A4) in the second subproblem to estimate the cost-to-go function $F_{t_{N-1}}$ and control policy $U_{t_{N-1}}^*$. Figure 3B4 illustrates possible state trajectories from a discrete storage level $x_{(t_{N-1})}^i$. Several trajectories are shown to emphasize that the precise trajectory depends on the particular outcome of $w_{(t_{N-1})}$. Solving for different states, we can again estimate a new cost-to-go function and control policy.

We repeat the process for every stage $t_j \in \{t_1, \dots, t_N\}$ until we arrive at the first stage. In each stage, the expected cost-to-go function $F_{t_j}(x)$ contains all necessary information about future stages. This function estimates the cost of future operations starting from state $x_{(t_j)}$ and assuming that optimal control policies are applied in remaining stages.

Figure 3B4. Discrete States and Sample State Trajectories for the Second-to-Last Subproblem



C. LIMITATIONS OF DISCRETE DYNAMIC PROGRAMMING

DDP estimates the cost-to-go by interpolating between nodes of a grid that spans possible states of a system, and the accuracy of estimates can be improved by using finer grids. However, grids that achieve sufficient accuracy may impose a tremendous computational burden. This is especially true when numerous state variables are required to model a system. We will see that the number of discrete states and, therefore, the effort required to solve a DDP problem grows exponentially with the number of state variables.

This exponential growth is known as DDP's "curse of dimensionality," and has led to a widely held view that practical application of DDP is limited to problems with only two or three state variables [Johnson *et al.*, 1993; Yakowitz, 1982]. As a result, DDP

as originally applied is unable to solve complex problems in which the state of a system is modeled by numerous variables [Yeh, 1985]. This has motivated efforts to find other stochastic methods (such as discussed in Chapter 3) capable of producing cautious control policies. More recently, this has also motivated efforts to improve DDP by using more accurate methods to approximate the cost-to-go function. With these recent improvements, DDP is able to solve problems with more state variables, and DDP can be an appropriate method for a wider variety of stochastic control problems.

1. Exponential Growth with State Dimension

Consider the single-reservoir problem illustrated in Figure 4C1. If there is but one state variable that represents the current amount of stored water, then we have a one-dimensional DDP. To approximate a cost-to-go function for each stage, we can identify the expected cost-to-go for each of Λ discrete states. Using DP, we find an optimal release policy and an expected cost-to-go function for each stage, working backwards in time. For this 1-D problem, the solution effort is proportional to the number of discrete states Λ and the number of stages N .

Now consider the n -dimensional problem illustrated in Figure 4C2 (e.g., a problem with multiple reservoirs or other state information). To approximate a cost-to-go function for each stage using Λ discrete values for each state variable, the total number of discrete states is Λ^n . The effort to solve this multi-dimensional problem is also proportional to the number of discrete states. This effort grows exponentially with the number of state variables, n , and the number discrete values, Λ , used to span each variable. In addition, the effort for each discrete state grows with the dimension of the problem since more effort is required to search for optimal control decisions.

In each stage, we conduct Λ^n searches (optimizations) to find optimal controls for each discrete state \mathbf{x}^i . As a result, the total effort per stage is

$$J = Z \Lambda^n \quad (4C1)$$

where n is the state-space dimension (i.e., number of state variables), Λ is a representative number of discrete values used to span each state variable, and Z is the effort to determine a solution for each of the Λ^n nodes. If we set $\Lambda = 20$ to develop sufficiently accurate control policies and cost functions, then a 2-D problem has 400 discrete states and a 6-D problem has 64,000,000 discrete states!

Furthermore, the effort Z increases with the dimension of a problem. This effort is a product of the time required to evaluate the total cost of equation (4A5) and the

number of evaluations required to find the solution of equation (4A6). In other words, the total effort is

$$J = Z_I Z_S Z_1 \Lambda^n \quad (4C2)$$

where Z_I is the time required to evaluate the total-cost of equation (4A5), Z_S is the number of evaluations required to search for the optimal solution of equation (4A6), and Z_1 is the number of searches and other overhead required for each node. Z_I increases with n and primarily depends on the effort to interpolate the cost-to-go function $F_{t_{j+1}}(\mathbf{y})$. Z_S also increases with n and depend on the solver employed to find the optimal control decisions. Typically, $Z_S \sim O(n^\alpha)$ where α depends on the search routine employed. For example, $\alpha = 0.5$ might apply for an efficient Newton-based search routine and $\alpha = 1.5$ might apply for a more robust simplex search routine (as we will see in Chapter 6). In addition to Z_I and Z_S , there may also be significant “overhead” effort required by the search routine and by other elements of the code used to implement the DDP algorithm and verify the results. In general, this overhead effort should become less significant part of the total effort with increasing dimension.

Figure 4C1. Discrete States for a 1-D Problem

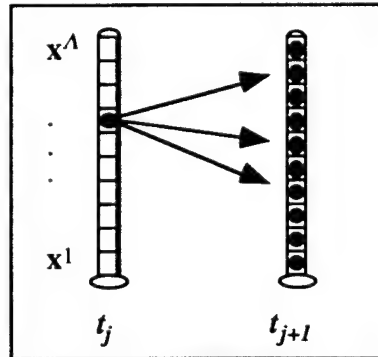
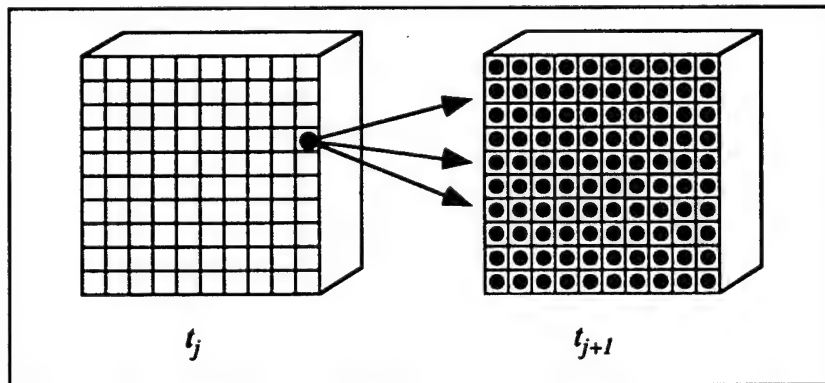


Figure 4C2. Discrete States for a Multidimensional Problem



2. Techniques to Reduce Exponential Growth

Equation (4C1) suggests that there are three approaches that we may use to reduce the exponential growth in effort of DDP. These are:

- (1) reduce n , the number of state variables
- (2) reduce Λ , the average number of discrete values for each state variable
- (3) reduce Z , the effort required to identify decisions for each discrete state

Each of these approaches is used in practice through the selection of practical models and optimization methods. However, each of these approaches also may require that we compromise the validity of a solution, through our use of inappropriately simple models. As a result, it is necessary to balance the accuracy of a model with the feasibility of solving the mathematical problem.

The first approach is to reduce the number of state variables n . Out of necessity, this approach is always employed when solving stochastic reservoir control problems. Almost all examples of DDP in the water resources literature have three or fewer state variables [Gal, 1979; Johnson *et al.*, 1991; Karamouz and Vasiliadis, 1992; Kelman *et al.*, 1990; Saad and Turgeon, 1988; Saad *et al.*, 1992], and, frequently, only one state variable is used. For example, a common technique in hydropower models is to aggregate the total stored water of multiple reservoirs into a single variable describing the total hydropower potential of stored water [Kelman *et al.*, 1990; Saad *et al.*, 1996; Turgeon, 1980; Turgeon, 1981; Valdes *et al.*, 1992]. Though sometimes appropriate, such extreme aggregation of system characteristics may, at other times, result in unrealistic system models and control solutions.

The second approach is to reduce the number of values Λ used to span a state variable. With only one or two state variables, reservoir control problems can be solved with relatively fine state-space grids. However, coarse grids are required when solving higher dimension problems [Esmaeil Beik and Yu, 1984], and coarse grids produce less accurate cost-to-go estimates and control policies that may be far from optimal. For example, Gal [1979] observed that DDP using coarse state-space grids may approximate cost-to-go functions worse than parameter iteration using a pre-determined class of functions. Also, because of coarse discretization and conditions that were closer to certainty equivalent, Karamouz and Houck [1987] observe that forecast-based methods identify more effective control decisions than DDP when applied to large reservoirs whose storage capacities exceed 50% of mean annual inflow. Figure 4C3 illustrates the effect that different state-space grids have on the accuracy of a cost function. Control

policies are especially sensitive to coarse grids because they should balance changes in current costs with changes in future costs. Changes in future costs are identified by cost-to-go function gradients, and estimation of these gradients is especially poor with coarse discretizations. *Esmail-Beik and Yu* [1984] provide an illustration of the trade-off in solution accuracy and computational effort for a single reservoir and correlated inflow. *Johnson et al.* [1993] provide an analysis the accuracy of cost functions and control policies for a range of grids applied to a four-reservoir test problem.

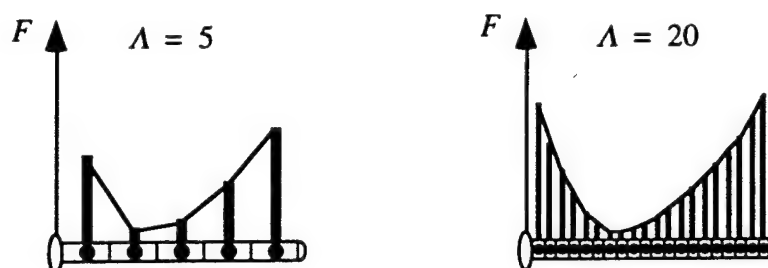
The third approach is to reduce the effort Z required to identify control decisions for each discrete state. This effort depends on mathematical properties of the model and on the search routine employed. An efficient routine applied to continuous and smooth functions can rapidly find optimal control decisions. For example, Newton-based search methods can converge if applied to smooth and convex cost functions. If these functions are not smooth or convex, we may need to employ more robust, but slower, search methods. Z also depends on the work required to evaluate a cost function with the general form given by equation (4A5). This is especially true if the interpolation method used to evaluate cost-to-go functions is complex. As a result, multilinear interpolation is traditionally used since it is the simplest interpolation method that produces a continuous estimate.

Besides these three basic approaches, there are a variety of other techniques that may significantly reduce the computational effort of some problems. These include: solving a sequence of problems with increasing difficulty, eliminating from consideration uninteresting areas of the state-space, partitioning the original problem into smaller separable problems, or creatively choosing variables used to model the system [*Johnson et al.*, 1988; *Johnson et al.*, 1993]. Nevertheless, when we model large-scale stochastic control problems, DDP becomes computationally infeasible. Until recently, we were instead required to use an approximate deterministic model [*Willis and Yeh*, 1987]. Though recent advances in stochastic control methods allow us to include greater detail in models, we still may require significant assumptions.

Because many problems cannot be simplified sufficiently and assumptions of other methods may be inappropriate, this thesis proposes DDP methods that reduce the number of discrete values, Λ , while maintaining interpolation accuracy. I accomplish this by using interpolation methods that are more accurate than traditional linear interpolation. Though this comes at the cost of a larger effort to evaluate interpolated values, these methods produce smooth and convex function approximations that allow application of efficient Newton-based search methods. What is more important is that we can significantly reduce Λ while preserving solution accuracy; and this dramatically reduces

the exponential growth of effort $J = Z \Lambda^n$. With these improvements, we can apply DDP to moderately complex reservoir management problems modeled with a greater number of state variables.

Figure 4C3. Cost Function Accuracy for Various State-Variable Discretizations



3. Cost-To-Go Interpolation Methods

Early applications of DDP used nearest-neighbor interpolation to estimate the cost-to-go by assigning the value evaluated at the nearest node of a state space grid [Buras, 1963]. No effort is required to evaluate the cost-to-go at intermediate states, and the problem can be solved using techniques applied to discrete-state dynamic programming models. However, nearest-neighbor interpolation produces models that assume inflows and releases are multiples of the discrete unit of storage [Bogle and O'Sullivan, 1979], and accurate solutions require a very fine discretization of the state space, typically about $\Lambda = 50$ to 100. As a result, DDP using nearest-neighbor interpolation can be accurately applied to problems with no more than one or two state variables. Other early applications of DDP included parametric DDP using pre-defined forms for the cost-to-go function [Bellman and Dreyfus, 1962; Gal, 1979].

Most current applications of DDP use multilinear interpolation. The effort required to evaluate the cost-to-go at intermediate states is modest, and the problem can be solved using a variety of robust search techniques. Multilinear interpolation significantly improves cost-to-go estimates, and accurate solutions can use a coarser discretization of the state space, typically about $\Lambda = 10$ to 20. Thus, DDP using multilinear interpolation can be accurately applied to problems with three or four state variables.

Recently, higher-order interpolation methods have been developed for DDP applications [Foufoula Georgiou and Kitanidis, 1988; Johnson et al., 1993]. Though these methods require additional effort to evaluate the cost-to-go at intermediate states, the problem may be solved with more efficient search techniques that find solutions quickly. These interpolation methods further improve cost-to-go estimates, and accurate

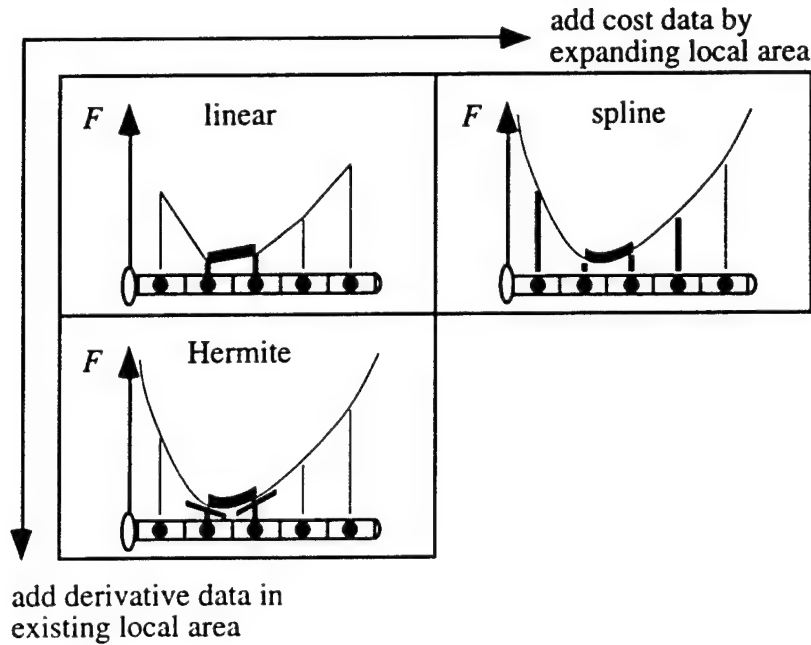
solutions can use a very coarse discretization of the state space, typically about $\Lambda = 3$ to 5. Though these methods have not seen significant application, DDP using these methods can be accurately applied to problems with five to seven state variables.

Although not technically not DDP, SDDP methods have recently been developed using another interpolation method of estimating. Instead of interpolation, SDDP uses cutting planes to identify a lower bound on the cost-to-go function [Pereira and Pinto, 1991]. The advantage of cutting planes is that they avoid the use of a regularly spaced grid and can be evaluated for an arbitrary state. In some cases, this may allow development accurate solutions with far fewer discrete states; though in other cases, the piecewise linear approximation of the cost-to-go function may be less accurate than high-order interpolation methods. However, a significant advantage of SDDP is that some estimate of a cost-to-go function can be developed, even for large-scale problems that cannot be addressed by DDP.

Traditional DDP fails to take advantage of the smoothness or, in the earliest applications, the continuity of the cost-to-go function. As a result, existing applications of DDP have required fine discretization of the state variables in order to achieve accurate estimates of the cost-to-go function and the control policies. For example, [Raman and Chandramouli, 1996] apply DDP to a stochastic reservoir control problem using from 11 to 51 discrete reservoir storage levels. In addition, it appears that they estimate the distribution of inflows using from 7 to 35 discrete inflow values. This high level of discretization does not impose a large computational burden on the solution of the single reservoir and inflow problem that they consider; however, for problems with multiple reservoirs and/or stochastic inputs, such a high level of discretization can make the solution intractable.

As discussed above, this thesis proposes DDP methods that reduce the number of discrete values, Λ , while maintaining interpolation accuracy. To accomplish this, I will use high-order interpolation methods that produce accurate cost-to-go estimates with coarse discretization of the state space. These high-order methods are local approximations that take advantage of additional information. Existing methods accomplish this using one of the following approaches: (1) using costs at a larger number of nodes (spline methods), or (2) using gradient information with cost information at corner nodes (Figure 4C4). In Chapter 5, we will use the second approach to develop high-order methods; and, in Chapter 6, we will apply these methods to problems with as many as seven state variables. This is a sufficient number of state variables to consider a variety of water-supply management problems without using the simplifying assumptions required by other stochastic dynamic control methods.

Figure 4C4. Comparison of Methods to Improve Interpolation Accuracy



D. MULTILINEAR INTERPOLATION

Traditionally, multilinear interpolation has been used to interpolate the cost-to-go for states not at the nodes of the grid. Multilinear interpolation is simple and is easier to compute than higher-order interpolation methods. Also, multilinear interpolation produces a much better estimate than that obtained using nearest-neighbor interpolation. In this thesis, I refer to DDP using multilinear interpolation as “multilinear DP.” This section develops multilinear DP to establish a methodology and notation for new methods presented in the next chapter. In later chapters, multilinear DP is also used to evaluate the improved performance of new methods.

1. Linear Interpolation in 1-D

Interpolation uses certain known information to estimate values of a continuous function [Davis, 1975, p17]. For example, linear interpolation uses known values $F(x^{(i)})$ to estimate the continuous function $F(x)$.

Interpolation produces a functional $\hat{F}(x)$ that we can use to represent the true function. Suppose we wish to approximate a 1-D function using values $F_i = F(x^{(i)})$ at nodes $x^{(i)}$, $i = 1, \dots, A$. Each adjacent pair of nodes $[x^{(i)}, x^{(i+1)}]$ defines an interval over which we can evaluate the approximating functional $\hat{F}_i(x)$ by a weighted sum

$$\hat{F}_i(x) = \phi_i(x) F_i + \phi_{i+1}(x) F_{i+1}$$

To preserve known values F_i and F_{i+1} , each weighting function $\phi_i(x)$ must equal 1 at node $x^{(i)}$ and 0 at the other node. The lowest-order polynomial functions that have these properties are

$$\phi_i(x) = (x - x^{(i+1)}) / \Delta x$$

$$\phi_{i+1}(x) = (x^{(i+1)} - x) / \Delta x$$

where $\Delta x = x^{(i+1)} - x^{(i)}$ is the length of interval $[x^{(i)}, x^{(i+1)}]$. Application of these linear weighting functions over each interval $i \in \{1, \dots, \Lambda-1\}$ produces a continuous functional $\hat{F}(x)$ that preserves function values at nodes.

2. Linear Interpolation in Multiple Dimensions

Suppose we wish to approximate a multi-dimensional function using values $F_i = F(\mathbf{x}^{(i)})$ at gridded nodes $\mathbf{x}^{(i)}$, $i = 1, \dots, \Lambda^n$. Because these nodes are gridded, they define corners of rectangular parallelepipeds or "hypercubes" that divide the function's domain. Figure 4D1 illustrates hypercubes of one, two, and three dimensions; and we see that each hypercube has 2^n corner nodes in an n -dimensional domain. Any value of \mathbf{x} can be identified as a point inside at least one hypercube (\mathbf{x} may lie in more than one hypercube when on a border).

Each set of corner nodes $\mathbf{x}^{(\gamma)}$, $\gamma = \gamma_1, \dots, \gamma_{2^n}$, define a subdomain over which we can calculate an approximating functional. The functional is a weighted sum of corner node values F_γ given by

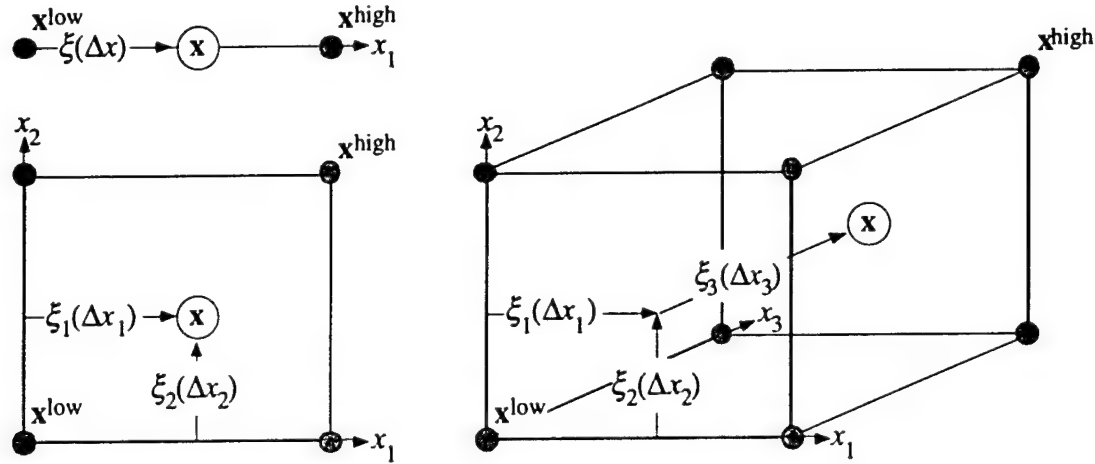
$$\hat{F}_i(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_{2^n}} \{ \phi_\gamma(\mathbf{x}) F_\gamma \} \quad (4D1)$$

where $\phi_\gamma(\mathbf{x})$ is the weight for corner node $\mathbf{x}^{(\gamma)}$. The i 'th subdomain is identified by its lower corner node $\mathbf{x}^{(i)} = \mathbf{x}^{(\gamma)}$. To preserve known values F_γ , each weighting function $\phi_\gamma(\mathbf{x})$, $\gamma = \gamma_1, \dots, \gamma_{2^n}$, must equal 1 at node $\mathbf{x}^{(\gamma)}$ and 0 at all other nodes. The lowest-order polynomial function that has these properties is

$$\phi_\gamma(\mathbf{x}) = \prod_{j=1}^n \{ |x_j - x_j^{(\gamma)}| / \Delta x_j \} \quad (4D2)$$

where Δx_j , $j = 1, \dots, n$, are the dimensions of the hypercube. Application of this multilinear weighting function to each hypercube produces a continuous functional $\hat{F}(\mathbf{x})$ that preserves function values at nodes.

Figure 4D1. Hypercubes of One, Two, and Three Dimensions



3. Local Coordinate System

Using notation for a local coordinate system, we can specify weighting functions that are simpler and more generic than those presented above. A local coordinate system is useful when interpolation depends only on nodes in a neighborhood of point \mathbf{x} . For example, linear interpolation depends only on corner nodes of the hypercube that immediately surrounds point \mathbf{x} . Notation for a local coordinate system will be especially useful in the next chapter to define other, more complex, interpolation methods.

When hypercubes are defined by a rectangular grid aligned to the coordinate system, we can observe that the corner nodes of each hypercube are identified by two nodes, \mathbf{x}^{low} and \mathbf{x}^{high} , at the lowest and highest corners defined by the coordinate system (i.e., closest to and farthest from the origin). Other corner nodes are an exhaustive combination of the coordinates of these two nodes. In Figure 4D2, for example, $\mathbf{x}^{\text{low}} = \mathbf{x}^{(2)}$ and $\mathbf{x}^{\text{high}} = \mathbf{x}^{(8)}$ define the shaded 2-D hypercube. Combining the coordinates of these nodes, we can identify the coordinates of the other corner nodes $\{\mathbf{x}^{(3)}, \mathbf{x}^{(7)}\}$ that are located at $\{(x_1^{\text{low}}, x_2^{\text{high}}), (x_1^{\text{high}}, x_2^{\text{low}})\}$. Also, $\Delta \mathbf{x} = \mathbf{x}^{\text{high}} - \mathbf{x}^{\text{low}}$ identifies the dimensions of the current hypercube.

Using \mathbf{x}^{low} and \mathbf{x}^{high} , we can transform the coordinates of any location \mathbf{x} into a local coordinate system. In each $j = 1, \dots, n$ dimension, we can transform a coordinate x_j into a local coordinate ξ_j by the equation

$$\xi_j = (x_j - x_j^{\text{low}}) / \Delta x_j$$

If \mathbf{x} is located in the hypercube identified by \mathbf{x}^{low} , then $0 \leq \xi \leq 1$ in all dimensions. For example, the corner nodes $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ are transformed as follows:

$$\begin{aligned}
\mathbf{x}^{\text{low}} = \mathbf{x}^{(\gamma_1)} &\Leftrightarrow \xi = [0,0,\dots,0] \\
\mathbf{x}^{(\gamma_2)} &\Leftrightarrow \xi = [1,0,\dots,0] \\
\mathbf{x}^{(\gamma_3)} &\Leftrightarrow \xi = [0,1,\dots,0] \\
&\dots \\
\mathbf{x}^{\text{high}} = \mathbf{x}^{(\gamma_n)} &\Leftrightarrow \xi = [1,1,\dots,1]
\end{aligned}$$

With this notation, we can combine equations (4D1) and (4D2) to compactly express the multilinear approximating functional. For the 2-D hypercube of Figure 4D2, this functional is

$$\hat{F}^{(i)}(\xi) = (1-\xi_2)[(1-\xi_1)F_2 + \xi_1 F_3] + \xi_2[(1-\xi_1)F_7 + \xi_1 F_8]$$

Point ξ in the local coordinate system is a transformation of \mathbf{x} , and $\{F_2, F_3, F_7, F_8\}$ are the known corner-node values.

The local coordinate system can also be used to develop other useful notation. The local coordinate system uses \mathbf{x}^{low} as the origin and specifies a normalized distance ξ between \mathbf{x} and corner-node \mathbf{x}^{low} . We can define the normalized distance between \mathbf{x} and any other corner-node $\mathbf{x}^{(\gamma)}$ as $\eta^{(\gamma)} = [\eta_1^{(\gamma)}, \dots, \eta_n^{(\gamma)}]^T$ where

$$\begin{aligned}
\eta_j^{(\gamma)} &= \xi_j & \text{if } x_j^{(\gamma)} &= x_j^{\text{low}} \\
\eta_j^{(\gamma)} &= (1 - \xi_j) & \text{if } x_j^{(\gamma)} &= x_j^{\text{high}}
\end{aligned}$$

For example, $\eta^{(\gamma)} = \xi^{(\gamma)}$ only if $\mathbf{x}^{(\gamma)} = \mathbf{x}^{\text{low}}$. Using this notation, we can transform the weighting-function of equation (4D2) to

$$\phi(\eta) = \prod_{j=1}^n (1-\eta_j)$$

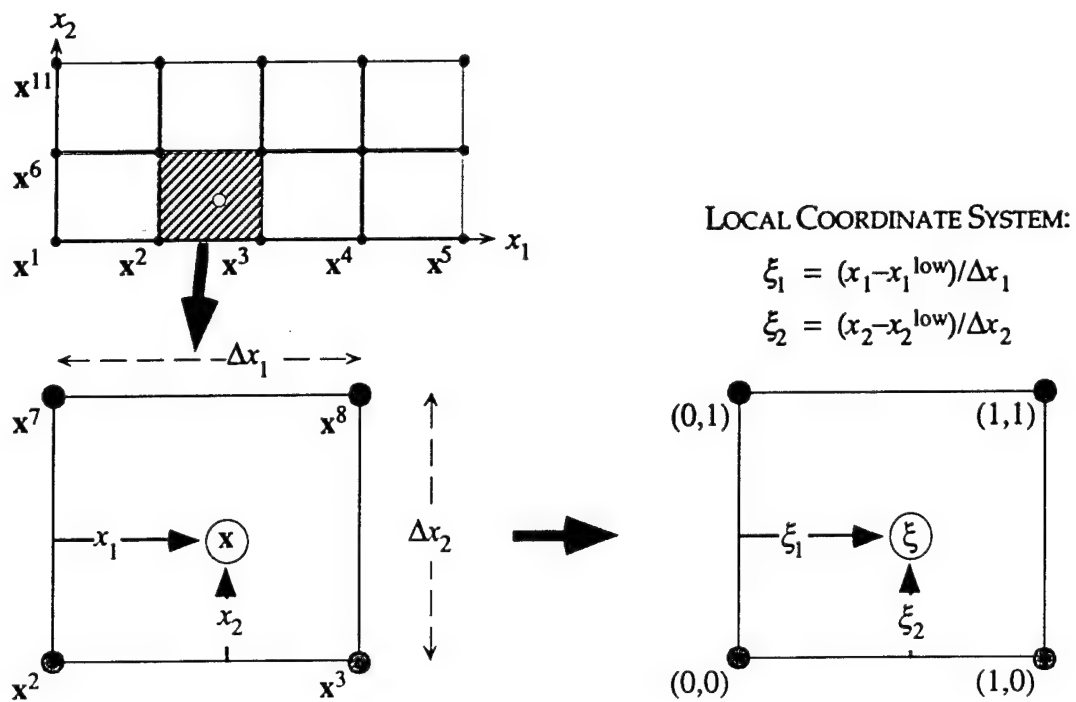
This notation allows us to express weighting functions that apply to all corner nodes $\mathbf{x}^{(\gamma)}$, $\gamma = \gamma_1, \dots, \gamma_n$.

In the next chapter, we will use this local-coordinate-system notation to develop approximating functionals that use not only the value F_γ at each corner-node $\mathbf{x}^{(\gamma)}$, but also the gradient \mathbf{G}_γ . For this, the approximating functional $\hat{F}^{(i)}(\mathbf{x})$ is a weighted sum

$$\hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_n} \{ \phi_\gamma(\mathbf{x}) F_\gamma + [\psi_\gamma(\mathbf{x})]^T \cdot \mathbf{G}_\gamma \} \quad (4D3)$$

$\psi_{\gamma j}(\mathbf{x})$ is the weight applied to corner-node gradient $G_{\gamma j} = dF(\mathbf{x})/dx_j$, $\mathbf{x} = \mathbf{x}^{(\gamma)}$. This is known as Hermite interpolation, and application of this to DDP is called "Gradient Dynamic Programming."

Figure 4D2. Local Coordinate System for Gridded 2-D Domain



CHAPTER 5.

NEW HERMITE INTERPOLATION METHODS

While DDP incorporates stochastic inputs with greater accuracy and fewer simplifying assumptions than is possible with other optimization methods, significant limitations prevent us from addressing large-scale reservoir management problems. This chapter presents new interpolation methods based on Hermite interpolation. Originally proposed for application to DDP by *Kitanidis* [1986], Hermite interpolation approximates cost-to-go functions with high-order accuracy and greatly reduces the exponential growth in computational effort. The methods presented in this chapter preserve the accuracy of the original interpolating functionals of *Foufoula-Georgiou and Kitanidis* [1988] while reducing the computational effort and providing smooth transitions between subdomains of the function.

Kitanidis and Foufoula-Georgiou [1987] coined the phrase “gradient dynamic programming” (GDP) to describe DDP using Hermite interpolation methods. While still more restrictive than other stochastic control methods, GDP allows solution of complex reservoir management problems without the simplifying assumptions of these other methods. This chapter concludes with an analysis of the effort required to implement the new GDP methods, and Chapter Six demonstrates their application to a range of multireservoir problems with as many as seven state variables.

A. CHARACTERISTICS OF AN EFFICIENT INTERPOLATION

The choice of interpolation method can significantly affect the solution accuracy and the computational effort required to solve DDP control problems. To solve DDP problems, interpolation methods should be (1) simple, allowing rapid evaluation, (2) smooth, allowing application of efficient search techniques that converge rapidly, and (3) accurate, allowing use of coarse grids. The first and second characteristics allow us to find optimal control decisions quickly (i.e., small Z). The third characteristic allows us to reduce the number of searches required (i.e., small Λ^n). All three decrease overall effort J and increase the number n of state variables that we are able to use in system modeling.

As a result, the choice of an interpolation method can significantly affect our ability to solve complex stochastic reservoir control problems.

Low-order polynomials have significant advantages over other functions in the construction of interpolating functionals used to approximate a cost-to-go function. Polynomials are simple to program and easy to evaluate. This is especially useful when used for multi-dimensional interpolation where approximating functionals are composed of many functions. Low-order polynomials may also yield approximations that are sufficiently smooth [Davis, 1975] to allow application of efficient quasi-Newton optimization routines.

For example, multilinear interpolation uses first-order polynomials that are simple and efficient. Unfortunately, multilinear interpolation is not very accurate on coarse grids, and it produces an approximation with undefined derivatives at subdomain boundaries. To find optimal control decisions, we must use fine grids and robust—but slow—search methods (instead of efficient, quasi-Newton search methods that require estimates of the gradient and Hessian of the cost-to-go function). Thus, DDP is severely limited in its ability to address complex problems when using multilinear interpolation.

Instead, we can use polynomials of somewhat higher-order that are more accurate and that produce smooth functionals. This is the approach taken by both spline methods [Johnson *et al.*, 1988; Johnson *et al.*, 1993] and Hermite methods [Foufoula Georgiou, 1991; Foufoula Georgiou and Kitanidis, 1988; Kitanidis, 1986; Kitanidis and Foufoula Georgiou, 1987]. In the case of spline methods, higher-order polynomials are used to incorporate cost-to-go information at nodes some distance beyond the surrounding hypercube. In the case of Hermite methods, higher-order polynomials are used to incorporate gradients or other derivative information, but only at nodes of the surrounding hypercube. In both cases, polynomials can be expressed as weights applied to node values, and interpolating functionals are linear combinations of these polynomials.

B. ADVANTAGES OF HERMITE INTERPOLATION

Hermite interpolation describes any method that approximates a function using both the function values and gradients at discrete nodes of a grid. For DDP applications, Hermite interpolation approximates the cost-to-go function with greater accuracy than spline methods and with significantly greater accuracy than multilinear interpolation [Johnson *et al.*, 1993]. This greater accuracy allows use of coarse state-space grids and dramatically reduces the exponential growth of effort required to solve DDP problems.

In addition, Hermite interpolation can be used to produce smooth cost-to-go approximations that allow application of rapid, quasi-Newton search methods.

However, Hermite interpolation requires additional computational effort, both to identify gradients at nodes and to incorporate gradient values in interpolating functionals. Fortunately, quasi-Newton search methods use an estimate of gradients to find optimal control decisions. This estimate is sufficiently accurate for direct use in Hermite interpolation, and we need only provide additional memory to store these values. Also, careful construction of interpolating functionals can minimize the effort required to incorporate gradient values. As we will see, the effort required for the first-order Hermite method presented here is only n -fold greater (where n is the number of state variables) than the effort for multilinear interpolation. Spline methods use cost-to-go values at a more extensive set of nodes than required by Hermite or multilinear interpolation, with an effort about 2^n -fold greater than the effort for multilinear interpolation.

Only recently has the benefit of higher-order interpolation methods been recognized. Previously, the effort required to apply Hermite interpolation to higher dimensions may have seemed excessive and likely to outweigh improvements in accuracy. Though Hermite interpolation has been applied to a variety of finite-element problems, including potential flow, boundary value, contaminant transport, and stress analysis [Foufoula Georgiou, 1991], these prior applications required only one, two, or three dimensions. Equivalent DDP problems had already been solved using traditional multilinear interpolation.

Kitanidis [1986] first proposed Hermite polynomials to estimate DDP cost-to-go functions. Subsequently, Hermite interpolation was applied to a one-reservoir problem [Kitanidis and Foufoula Georgiou, 1987] and a four-reservoir problem [Foufoula Georgiou and Kitanidis, 1988]. Spline methods were introduced by Johnson *et al.* [1988; 1993] to avoid the use of gradients while producing smooth interpolations that also permit application of rapid, quasi-Newton search methods. They compared spline methods with the Hermite method of Foufoula-Georgiou and Kitanidis, concluding that performance of the two methods was roughly equal. In application to a series of test problems with up to five dimensions, they found that their spline method benefited from higher-order smoothness and more rapid convergence of search methods and that Gradient DP benefited from higher accuracy.

This chapter presents new interpolation methods based on Hermite interpolation. These methods maintain the high-order accuracy of the original Hermite methods of Foufoula-Georgiou and Kitanidis while achieving higher-order smoothness, allowing

more rapid convergence of search methods. Also, the computational effort required to evaluate interpolating functionals is significantly reduced.

C. CHARACTERISTICS OF WEIGHTING FUNCTIONS

The interpolation methods presented in this thesis assume that discrete states $\mathbf{x}^{(i)}$ are located at the nodes of a grid that spans all possible states of a system. There are V^n nodes for an n -dimensional grid, where V is the average number discrete values for each state variable. At each node, we evaluate a value F_i and a gradient $\mathbf{G}_i = [G_{i,1}, \dots, G_{i,n}]^T$, where $G_{i,j} = dF/dx_j^{(i)}$ is the derivative in the j 'th dimension. This grid divides the domain of a cost-to-go function into subdomains with corner nodes $\mathbf{x}^{(\gamma_1)}, \dots, \mathbf{x}^{(\gamma_n)}$.

Over each subdomain, we interpolate the cost-to-go value and gradient using a weighted sum of corner-node values and gradients. To preserve the cost-to-go values and gradients and to produce continuous and smooth approximating functionals, each weighting function $\phi_\gamma(\mathbf{x})$ and $\psi_{\gamma,j}(\mathbf{x})$, $\gamma = \gamma_1, \dots, \gamma_n$, $j = 1, \dots, n$, must satisfy a number of requirements. Continuous and smooth functionals allow us to use of efficient quasi-Newton search methods. Also, smooth interpolating functionals accurately represent true cost-to-go functions because stochastic inputs result in a distribution of future states that smooth out the expected cost of future operations.

1. Requirements to Preserve Node Values and Gradients in One Dimension

Suppose we wish to interpolate a one-dimensional cost-to-go function $F(x)$ using function values $F_i = F(x^{(i)})$ and derivatives $G_i = dF/dx^{(i)}$ at nodes $x^{(i)}$, $i = 1, \dots, \Lambda$. Over an interval $[x^{(i)}, x^{(i+1)}]$, the interpolating functional is given by the weighted sum

$$\hat{F}^{(i)}(x) = \phi_i(x) F_i + \psi_i(x) G_i + \phi_{i+1}(x) F_{i+1} + \psi_{i+1}(x) G_{i+1}$$

If this functional is to preserve the cost-to-go value F_i , the weight $\phi_i(x)$ must equal one and all other weights must vanish when $x = x^{(i)}$. Likewise, if this functional is to preserve the cost-to-go value F_{i+1} , the weight $\phi_{i+1}(x)$ must equal one and all other weights must vanish when $x = x^{(i+1)}$. Over the same interval, the gradient of the interpolating functional is given by the weighted sum

$$\hat{G}^{(i)}(x) = \frac{\partial \phi_i}{\partial x} F_i + \frac{\partial \psi_i}{\partial x} G_i + \frac{\partial \phi_{i+1}}{\partial x} F_{i+1} + \frac{\partial \psi_{i+1}}{\partial x} G_{i+1}$$

If this functional is to preserve the cost-to-go gradient G_i , the weight $\partial \psi_i / \partial x$ must equal one and all other weights must vanish when $x = x^{(i)}$. Likewise, if this functional is to

preserve the cost-to-go gradient G_{i+1} , the weight $\partial\psi_{i+1}/\partial x$ must equal one and all other weights must vanish when $x = x^{(i+1)}$. Table 5C1 summarizes these weighting-function requirements. There are four constraints on each weighting function that can be satisfied by third-order polynomial functions. Also, the resulting interpolating functionals are continuous and smooth across all intervals.

Table 5C1. Weighting-Function Requirements for Interpolation in One-Dimension

	Zero'th-Order Value	First Derivatives
$\phi_i(x)$	1 at node $x^{(i)}$, 0 at node $x^{(i+1)}$	0 at both nodes
$\psi_i(x)$	0 at both nodes	1 at node $x^{(i)}$, 0 at node $x^{(i+1)}$

2. Requirements to Preserve Node Values and Gradients in Multiple Dimensions

Similarly, suppose we wish to interpolate a multi-dimensional cost-to-go function $F(\mathbf{x})$ using function values $F_i = F(\mathbf{x}^{(i)})$ and gradients $\mathbf{G}_i = dF/d\mathbf{x}^{(i)}$ at nodes $\mathbf{x}^{(i)}$, $i = 1, \dots, \Lambda^n$. There are 2^n cost-to-go values and $n 2^n$ derivatives in each n -dimensional hypercube. Over an n -dimensional hypercube with lower corner node $\mathbf{x}^{(i)}$, the interpolating functional is given by the weighted sum of equation (4D3)

$$\hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_i}^{\gamma^n} \{ \phi_\gamma(\mathbf{x}) F_\gamma + [\psi_\gamma(\mathbf{x})]^T \cdot \mathbf{G}_\gamma \} \quad (5C1)$$

If this functional is to preserve a cost-to-go value F_γ , the weight $\phi_\gamma(\mathbf{x})$ must equal one and all other weights must vanish when $\mathbf{x} = \mathbf{x}^{(\gamma)}$. Over the same hypercube, the gradient of the interpolating functional is given by the weighted sum

$$\hat{\mathbf{G}}^{(i)}(\mathbf{x}) = \frac{\partial \hat{F}^{(i)}}{\partial \mathbf{x}} = \sum_{\gamma=\gamma_i}^{\gamma^n} \left\{ \frac{\partial \phi_\gamma}{\partial \mathbf{x}} F_\gamma + \left[\frac{\partial \psi_\gamma}{\partial \mathbf{x}} \right]^T \cdot \mathbf{G}_\gamma \right\} \quad (5C2)$$

If this functional is to preserve a cost-to-go derivative $G_{\gamma j}$, the weight $\partial\psi_{i,j}/\partial x_j$ must equal one and all other weights must vanish when $\mathbf{x} = \mathbf{x}^{(\gamma)}$. Table 5C2 summarizes these weighting-function requirements. Using notation of the local coordinate system presented in Section 4D3, $\mathbf{x} = \mathbf{x}^{(\gamma)}$ when $\eta = 0$.

In an n -dimensional problem, there are potentially $(1+n) 2^n$ constraints on each weighting function $\phi_\gamma(\mathbf{x})$ and $\psi_{\gamma j}(\mathbf{x})$. However, many of these constraints are redundant since a constraint that causes a weight to vanish for one node and dimension also causes it to vanish for other nodes and other dimensions. For example, we can satisfy the requirements of Table 5C2 by weights that are an n -fold product of functions. Each function of this product contains a single state variable x_j and satisfies the requirements

of Table 5C1. Thus, each weight is an n -fold product of third-order polynomials (or higher), or a $3n$ -fold product of linear terms. However, unlike the 1-D case, weighting functions that satisfy the requirements of Table 5C2 may produce interpolating functionals that are discontinuous and not smooth. Therefore, requirements for continuous and smooth interpolation go beyond those that preserve cost-to-go values and gradients at nodes.

Table 5C2. Weighting-Function Requirements for Interpolation in Multiple Dimensions

	Zero'th-Order Value	First Derivative in dimension x_s
$\phi_\gamma(\mathbf{x})$	1 at node $\mathbf{x}^{(\gamma)}$, 0 at other nodes	0 at all nodes
$\psi_{\gamma j}(\mathbf{x})$	0 at all nodes	1 at node $\mathbf{x}^{(\gamma)}$, $s = j$, 0 at node $\mathbf{x}^{(\gamma)}$, $s \neq j$, 0 at other nodes.

3. Additional Requirements to Produce Continuous and Smooth Interpolating Functionals in Multiple Dimensions

To ensure continuity and smoothness between hypercubes of higher dimension, we must ensure that weighting functions satisfy additional requirements beyond those that preserve cost-to-go values and gradients at nodes. We must also ensure that weighting functions for adjacent hypercubes produce the same cost-to-go values and gradients along the shared boundary. This means that the interpolating functionals on both sides of a shared boarder must converge to the same equation.

Two conditions must be satisfied to ensure this convergence. The first condition is that only shared nodes can be used to evaluate a boundary, and weights applied to nodes not on the boundary must vanish. Otherwise, nodes not on the boundary can influence estimated values and gradients along the boundary. The second condition is that unless other properties of hypercubes (such as geometry) are shared, they cannot influence estimated values and gradients along boundaries.

To satisfy these conditions on a regular grid (i.e., a grid with discretization intervals of equal length), we apply two additional constraints to weighting functions. The first constraint requires that we use interpolating functionals with consistent form for each hypercube. Unfortunately, this prevents us from selecting among different interpolations (such as the interpolations of *Foufoula-Georgiou* [1991] that can be used to preserve convexity) based on the local character of a cost-to-go function. The second

constraint is that weights vanish when applied to unshared node values. This requires that weights vanish at any location where $\eta_j = 1$ in any dimension $j = 1, \dots, n$ (Table 5C3).

If the domain is divided by an irregular grid with discretization intervals of variable length, the above constraints will be insufficient to guarantee continuity and smoothness. An example of an irregular grid is an adaptive grid that uses finer discretization intervals only where needed. With an irregular grid, we must be careful to avoid terms that depend on hypercube dimensions. In particular, if a weight depends on an unequal discretization interval Δx_k , this weight must vanish along hypercube boundaries both where $\eta_k = 0$ and where $\eta_k = 1$ (Table 5C4).

Table 5C3. Weighting-Function Requirements for Continuity and Smoothness on a Regular Grid

	Zero'th-Order Value	First Derivative in dimension x_s
$\phi(\eta)$	1 where $\eta = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$	0 where $\eta = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$
$\psi_j(\eta)$	0 where $\eta = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$	1 where $\eta = 0, s = j$, 0 where $\eta = 0, s \neq j$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$

Table 5C4. Weighting-Function Requirements for Continuity and Smoothness on an Irregular Grid

	Zero'th-Order Value	First Derivative in dimension x_s
$\phi(\eta)$	1 where $\eta = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$	0 where $\eta_s = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$
$\psi_j(\eta)$	0 where $\eta_j = 0$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$	1 where $\eta = 0, s = j$, 0 where $\eta_j = 0$ and $\eta_s = 0, s \neq j$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$

D. ORIGINAL HERMITE INTERPOLATION METHOD

This section presents the multi-dimensional Hermite interpolation presented by *Kitanidis* [1986] and applied to DDP by *Foufoula-Georgiou and Kitanidis* [1988]. This first application of Hermite interpolation to a DDP problem produced accurate estimates of the cost-to-go and demonstrated an ability to solve stochastic dynamic control problems with a greater number of state variables.

1. The Weighting Functions

This interpolation uses the polynomial weighting functions [Foufoula Georgiou and Kitanidis, 1988, Appendices A and B]:

$$\phi(\eta) = R P \quad (5D1)$$

$$\psi_j(\eta) = \eta_j (1 - \eta_j) P \quad (5D2)$$

The terms R and P are given by the equations

$$R = 1 + \sum_{k=1}^n \eta_k (1 - 2\eta_k) \quad (5D3)$$

$$P = \prod_{k=1}^n \{1 - \eta_k\} \quad (5D4)$$

These weighing functions produce accurate cost-to-go approximating functionals $\hat{F}(\mathbf{x})$ even when using coarse grids.

However, these weighing functions do not produce smooth cost-to-go approximating functionals (i.e., the gradient estimate $\hat{\mathbf{G}}(\mathbf{x})$ is not continuous). Equations (5D1) and (5D2) do satisfy the requirements of Table 5C3 because their derivatives do not vanish for some $\eta_k = 1$. The derivatives of equations (5D1) and (5D2) are

$$\frac{d\phi}{dx_s} = \frac{\partial \eta_s}{\partial x_s} [(1 - 4\eta_s) P - R P_{(s)}]$$

$$\frac{d\psi_j}{dx_s} = (1 - \eta_j) (1 - 3\eta_j) P_{(s)}, \quad s = j$$

$$\frac{d\psi_j}{dx_s} = \frac{\partial \eta_s}{\partial x_s} \left(\frac{\partial \eta_j}{\partial x_j} \right)^{-1} \eta_j (\eta_j - 1) P_{(s)}, \quad s \neq j$$

The term $P_{(s)}$ is the previous product P exempting terms indicated by the subscript:

$$P_{(s)} = \prod_{k=1, k \neq s}^n \{1 - \eta_k\}$$

When $\eta_s = 1$, the derivatives produce weights

$$\frac{d\phi}{dx_s} = \frac{\partial \eta_s}{\partial x_s} P_{(s)} \sum_{k=1, k \neq s}^n \{ \eta_k (1 - 2\eta_k) \}$$

$$\frac{d\psi_j}{dx_s} = \frac{\partial \eta_s}{\partial x_s} \left(\frac{\partial \eta_j}{\partial x_j} \right)^{-1} \eta_j (\eta_j - 1) P_{(s)}, \quad s \neq j$$

The derivatives are not equal to zero except for values of η such as at the corner nodes or when $\eta_k = 1, k \neq s$. As a result, weights do not always vanish when applied to nodes not on a shared border. Nevertheless, equations (5D1) and (5D2) do produce continuous cost-to-go approximations since the weights given by equations (5D1) and (5D2) vanish if $\eta_k = 1$ for any $k = 1, \dots, n$.

Discontinuous gradients between hypercubes can complicate application of Newton-based search methods and may require that we use a more robust, but slower, search method. For example, *Johnson et al.* [1993] applied a quasi-Newton search method to their test of Gradient DP by restarting the search whenever a hypercube boundary was crossed.

2. Analysis of the Original Hermite Interpolation Method

Johnson et al. [1993] compared the efficiency of Hermite and spline methods with linear interpolation, demonstrating that the improved accuracy of higher-order methods more than compensates for the increased complexity of the interpolation. They observed a significant reduction in the computational effort required to achieve comparable levels of accuracy. For example, computational effort was reduced by a factor of 250 for a four-reservoir control problem. While the savings for spline and Hermite methods were comparable, they evaluated the spline method to be slightly more efficient. In spite of the superior accuracy of Hermite interpolation, they judged this insufficient to overcome the effort required to calculate both function values and gradients at nodes.

However, the original Hermite method does not produce continuous derivatives, and the quasi-Newton search method is not as effective at searching for the best control decisions or at estimating gradients. Also, it appears that Hermite interpolation may gain the advantage over spline methods at higher dimension than considered by *Johnson et al.* [1993]. Hermite interpolation uses values only at nodes on the boarder of a subdomain, or a total of $(1+n)2^n$ values. The spline method of *Johnson et al.* [1993] uses 4^n node values over a larger region. Indeed, they observe that after four state variables, the efficiency of the spline method does not improve rapidly in comparison with multilinear interpolation; multilinear interpolation, like Hermite interpolation, uses values only at immediately adjacent nodes, or a total of 2^n values.

The following three sections present new interpolation methods that improve on the original method. The first is a method that uses first-derivatives in what I call the

“first order” Hermite method to distinguish it from methods that use higher derivatives. Following this is a method that produces continuous second derivatives. A third section presents a “second order” method that uses some second derivatives. This third new method produces a more accurate interpolation that is better at preserving the convexity of the true cost-to-go function. All of these methods take advantage of the high-order accuracy of Hermite interpolation while producing a smooth interpolation to allow better implementation of quasi-Newton search methods. Also, these methods are easier to evaluate than the original Hermite interpolation method.

E. NEW FIRST-ORDER HERMITE INTERPOLATION METHOD

This section presents multidimensional interpolation method that uses cost-to-go values and first-derivatives to produce smooth and continuous approximating functionals. Multidimensional interpolation may require complex interpolating functionals of high dimension. This results from the exponential growth in the number of node values that must be preserved. This also results from additional requirements used to enforce continuity, smoothness, or other desirable qualities of an interpolating functional.

The weights used in this method are the lowest-order polynomial functions that satisfy the requirements of Table 5C4. Low-order functions reduce the likelihood that an interpolating functional will oscillate. In addition, low-order functions increase the likelihood that an interpolating functional will be strictly convex (or concave) when the true cost-to-go function strictly convex (or concave). Because these polynomials satisfy the requirements of Table 5C4, the interpolation can be applied to grids with any arbitrary discretization interval.

This method is used to produce a highly efficient numerical code that is contained in an appendix to this dissertation. As discussed earlier, efficiency is provided by interpolation methods that have higher-order accuracy and that are smooth. However, efficiency is also provided by interpolation methods that are easy to evaluate. For example, the traditional multilinear interpolation is exceptionally easy to evaluate, and this accounts for much of its popularity in DDP. Likewise, spline and Hermite interpolation methods use low-order polynomials and, thus, are also relatively easy to evaluate.

This interpolation method improves upon the original multidimensional Hermite interpolation method of *Kitanidis* [1986] by ensuring continuity of first derivatives over the entire domain while also reducing the required computational effort. Continuous first

derivatives allow us to apply quasi-Newton search methods that also give us gradient information for little or no extra computational effort, though it does increase storage by a factor of $n+1$ for an n -dimensional problem. The derivation in this section parallels the development of multilinear interpolation in Chapter 4.

1. Hermite Interpolation in One Dimension

To produce weighting functions that are simple to evaluate, we choose the lowest order polynomial functions that satisfy the requirements of Table 5C1. In a 1-D problem, there are four constraints on each weighting function. To satisfy four constraints, each weighting function must be a third-order or higher polynomial.

Third-order polynomial functions that satisfy these constraints were identified by *Kitanidis and Fouloula-Georgiou* [1987] as:

$$\phi_i(x) = [2(x-x^{(i)} + \Delta x) (x^{(i+1)}-x)^2 / (\Delta x)^3]$$

$$\phi_{i+1}(x) = [2(x^{(i+1)}-x) + \Delta x] (x-x^{(i)})^2 / (\Delta x)^3]$$

$$\psi_i(x) = (x-x^{(i)}) (x^{(i+1)} - x)^2 / (\Delta x)^2]$$

$$\psi_{i+1}(x) = - (x^{(i+1)}-x) (x-x^{(i)})^2 / (\Delta x)^2]$$

Because these functions preserve both cost-to-go values and gradients at nodes, the resulting 1-D interpolation is both continuous and smooth. Figure 5E1 compares 1-D linear and Hermite interpolation for an example cost-to-go function $F(x) = 1/x$. The first two plots use nodes $x^{(i)} \in \{1,4\}$, and the second two plots use nodes $x^{(i)} \in \{1,2,4\}$. These plots illustrate the higher-order accuracy of Hermite methods: with few nodes, the Hermite interpolation produces an accurate and smooth approximation of cost-to-go values and gradients.

We can simplify expression of the 1-D weighting functions using the local coordinate system notation of Section 4D3. Using this notation, the weighting functions are given by

$$\phi(\eta) = (1+2\eta) (1-\eta)^2$$

$$\psi(\eta) = \eta (1-\eta)^2 (d\eta/dx)^{-1}$$

and their derivatives are

$$\frac{d\phi}{d\eta} = -6\eta (1-\eta) \frac{d\eta}{dx}$$

$$\frac{d\psi}{d\eta} = (1-3\eta)(1-\eta)$$

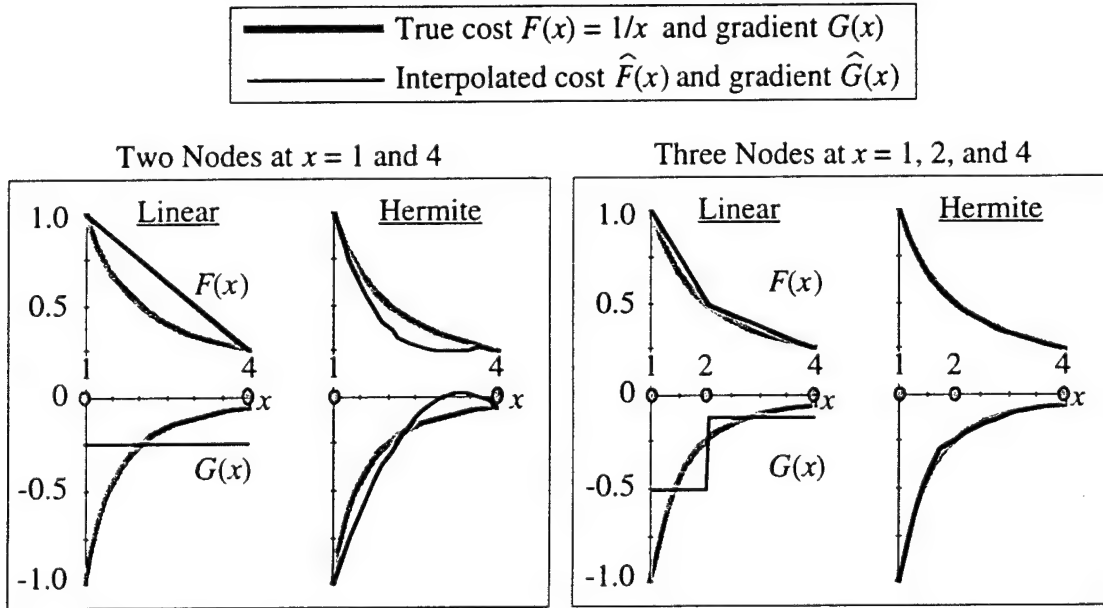
The resulting 1-D cost-to-go value and gradient functionals over any interval $[x^{(i)}, x^{(i+1)}]$ are

$$\hat{F}(\xi) = (1+2\xi)(1-\xi)^2 F_i + (3-2\xi)\xi^2 F_{i+1} + \xi(1-\xi) [(1-\xi)G_i - \xi G_{i+1}] (\Delta x) \quad (5E1)$$

$$\hat{G}(\xi) = \frac{d\hat{F}}{d\xi} = 6\xi(1-\xi) \frac{F_{i+1}-F_i}{\Delta x} + (1-3\xi)(1-\xi) G_i - (2-3\xi)\xi G_{i+1} \quad (5E2)$$

Note that $d\xi/dx = 1/\Delta x$.

Figure 5E1. 1-D Linear and First-Order Hermite Interpolation of the Function $F(x) = x^{-1}$



2. Hermite Interpolation in Multiple Dimensions

To produce multi-dimensional weighting functions that are simple to evaluate, we choose the lowest-order polynomial functions that satisfy the requirements of Table 5C4. Unlike the multi-dimensional requirements of Table 5C2 that are enforced only at corner nodes, most of the requirements of Table 5C4 are enforced along entire boundaries. More specifically, the weights $\phi(\eta)$ and $\psi(\eta)$ and their first derivatives must be zero for any $\eta_k = 1$ and even for some $\eta_k = 0$. In an n -dimensional problem, there are $(1+n) 2^n$ cost-to-go values and first-derivatives, so there are potentially $(1+n) 2^n$ constraints on each weighting function $\phi_j(\mathbf{x})$ and $\psi_{j,j}(\mathbf{x})$. However, many of these constraints are redundant since a constraint that causes a weight to vanish for one node and dimension

also causes it to vanish for other nodes and other dimensions. As suggested in Section 5C2, we can identify each weighting function as an n -fold product of 1-D functions. In each dimension there are either three or four constraints that can be satisfied by second-order or third-order polynomials. If we use only regular (i.e., evenly spaced) state-space grids, then we may be able to reduce the order of some polynomials.

Weighting functions that are an n -fold product of 1-D functions and that produce continuous first derivatives for either regular and irregular grids are

$$\phi(\eta) = P = \prod_{k=1}^n (1+2\eta_k)(1-\eta_k)^2$$

$$\psi_j(\eta) = Q_j P_{(j)} (d\eta_j/dx_j)^{-1}$$

where

$$Q_j = \eta_j(1-\eta_j)^2$$

$$P_{(j)} = \prod_{k=1, k \neq j}^n (1+2\eta_k)(1-\eta_k)^2$$

The first derivatives are

$$\frac{d\phi}{dx_s} = -6\eta_s(1-\eta_s)P_{(s)} \frac{d\eta_s}{dx_s}$$

$$\frac{d\psi_j}{dx_s} = (1-3\eta_s)(1-\eta_s)P_{(s)}, \quad s = j$$

$$\frac{d\psi_j}{dx_s} = -6\eta_s(1-\eta_s)Q_j P_{(s,j)} (d\eta_j/dx_j)^{-1} \frac{d\eta_s}{dx_s}, \quad s \neq j$$

Because these satisfy the requirements of Table 5C4 (Figures 5E2 through 5E5), the resulting multidimensional interpolation is both continuous and smooth. Figure 5E6 compares 2-D linear and Hermite interpolation for an example cost-to-go function $F(\mathbf{x}) = (x_1 x_2)^{-1}$. The first two plots use nodes formed from the set of coordinates $x_j^{(i)} \in \{1,4\}$, $j = 1,2$; and the second two plots use nodes formed from the set of coordinates $x_j^{(i)} \in \{1,2,4\}$.

In contrast to the 1-D case, these weighting functions are not the lowest-order polynomials that guarantee continuous first derivatives. As mentioned, the requirements of Table 2C4 could be satisfied with second-order polynomials in some cases. However, experimentation with weights composed of mixed second-order and third-order polynomials did not produce interpolating functionals with noticeably lower curvature. Also, these mixed-order weights were not able to preserve convexity and they required

more effort to evaluate. If grids are regular in some or all dimensions, the requirements of Table 2C3 could be applied to further reduce the order of polynomials. I did not experiment with weights composed of such lower-order polynomials, and this presents a possible area for further refinements.

To demonstrate the application of these weighting functions, we can state the closed-form approximating functional for 2-D interpolation. The 2-D approximating functional is

$$\hat{F}(x) = f_0 + f_1 + f_2$$

$$\text{where } f_0 = P_{(2)}[P_{(1)}F_{\eta_1} + P'_{(1)}F_{\eta_2}] + P'_{(2)}[P_{(1)}F_{\eta_2} + P'_{(1)}F_{\eta_3}]$$

$$f_1 = Q_1[P_{(1)}G_{\eta_1,1} + P'_{(1)}G_{\eta_2,1}]\Delta x_1 - Q'_1[P_{(1)}G_{\eta_2,1} + P'_{(1)}G_{\eta_3,1}]\Delta x_1$$

$$f_2 = P_{(2)}[Q_2G_{\eta_1,2} - Q'_2G_{\eta_2,2}]\Delta x_2 - P'_{(2)}[Q_2G_{\eta_2,2} + Q'_2G_{\eta_3,2}]\Delta x_2$$

Consistent with previous definitions,

$$\text{when } \eta_1 = \xi_1: \quad P_{(2)} = (1+2\xi_1)(1-\xi_1)^2$$

$$Q_1 = \xi_1(1-\xi_1)^2$$

$$\text{and when } \eta_1 = 1-\xi_1: \quad P'_{(2)} = (3-2\xi_1)\xi_1^2$$

$$Q'_1 = (1-\xi_1)\xi_1^2$$

$P_{(1)}$, Q_2 , $P'_{(1)}$, and Q'_2 are defined similarly. The derivatives of the approximating functional are

$$\frac{d\hat{F}}{dx_1} = \frac{df_0}{dx_1} + \frac{df_1}{dx_1} + \frac{df_2}{dx_1}$$

$$\text{where } \frac{df_0}{dx_1} = (6)\xi_1(1-\xi_1)[P_{(1)}\frac{F_{\eta_2}-F_{\eta_1}}{\Delta x} + P'_{(1)}\frac{F_{\eta_3}-F_{\eta_2}}{\Delta x}]$$

$$\frac{df_1}{dx_1} = (1-3\xi_1)(1-\xi_1)[P_{(1)}G_{\eta_1,1} + P'_{(1)}G_{\eta_2,1}] - (2-3\xi_1)\xi_1[P_{(1)}G_{\eta_2,1} + P'_{(1)}G_{\eta_3,1}]$$

$$\frac{df_2}{dx_1} = (6)\xi_1(1-\xi_1)[Q_2(G_{\eta_2,2}-G_{\eta_1,2}) - Q'_2(G_{\eta_3,2}-G_{\eta_2,2})]\frac{\Delta x_2}{\Delta x_1}$$

The equation for $d\hat{F}/dx_2$ parallels that for $d\hat{F}/dx_1$.

Figure 5E2. 2-D Weighting Function $\phi(\eta)$ Applied at a Node

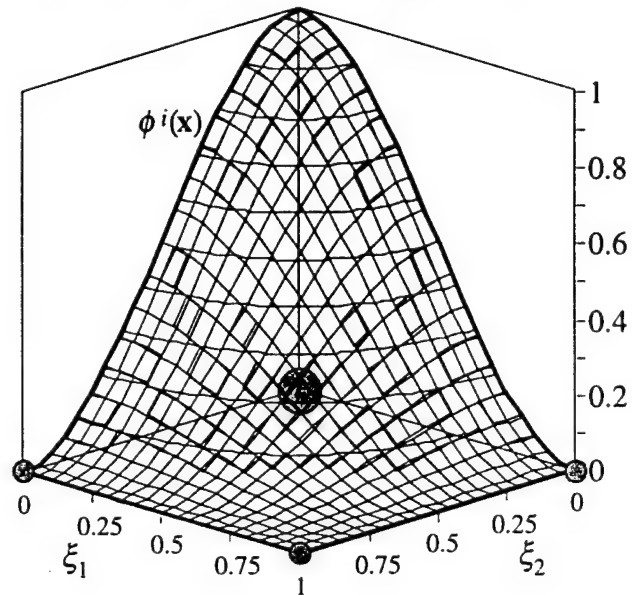


Figure 5E3. 2-D Derivatives of Weighting Function $\phi(\eta)$ Applied at a Node

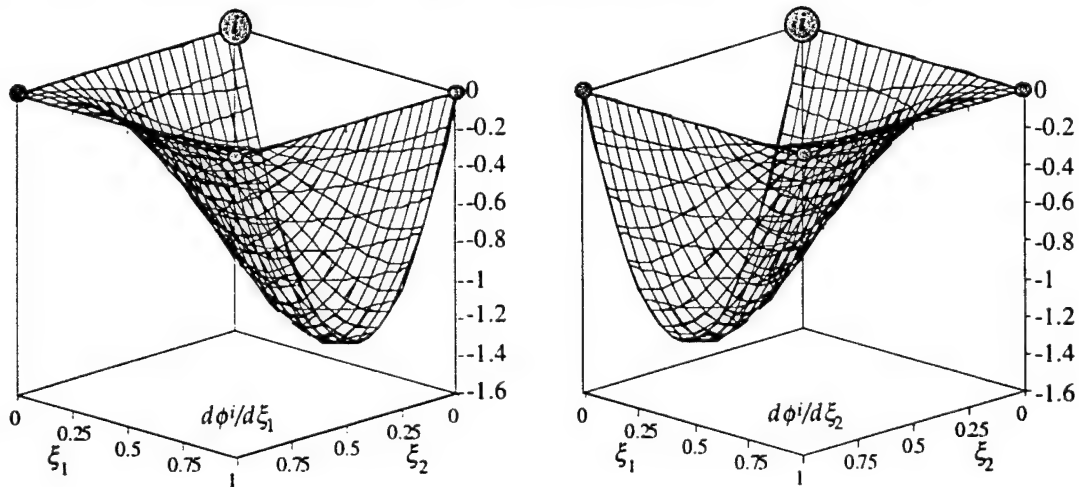


Figure 5E4. 2-D Weighting Function $\psi(\eta)$ Applied at a Node

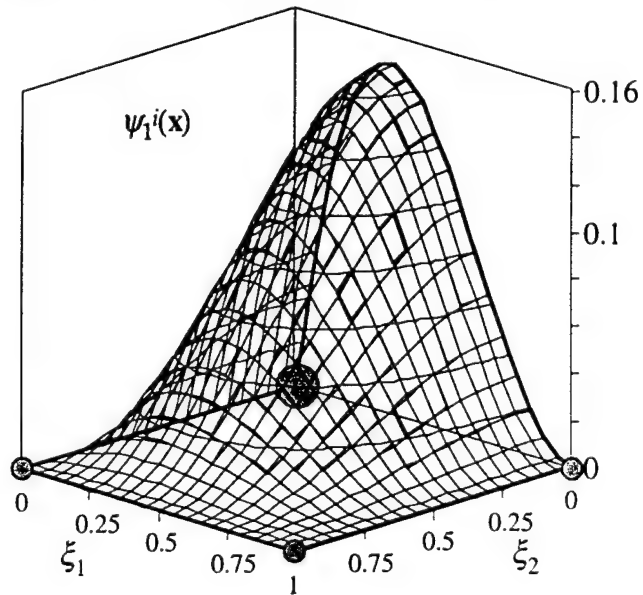


Figure 5E5. 2-D Derivatives of Weighting Function $\psi(\eta)$ Applied at a Node

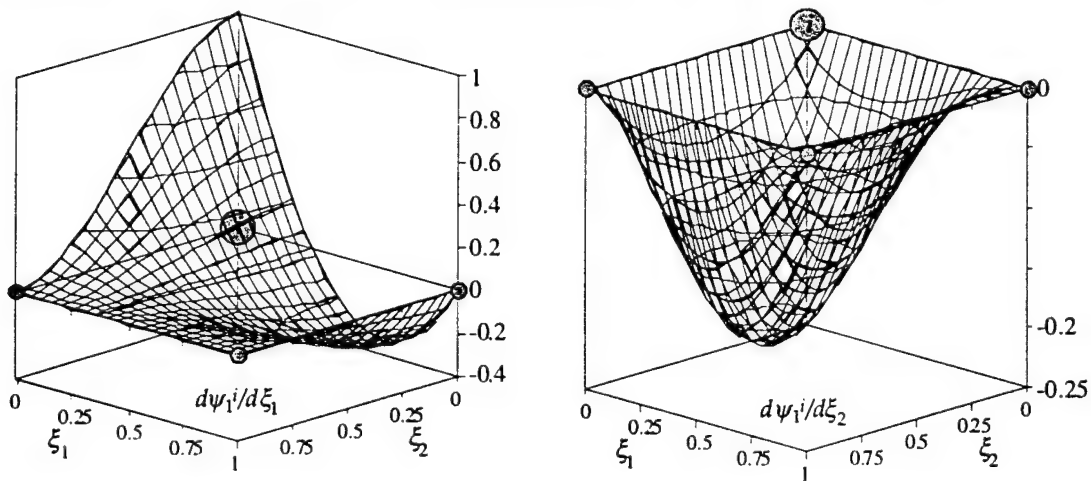
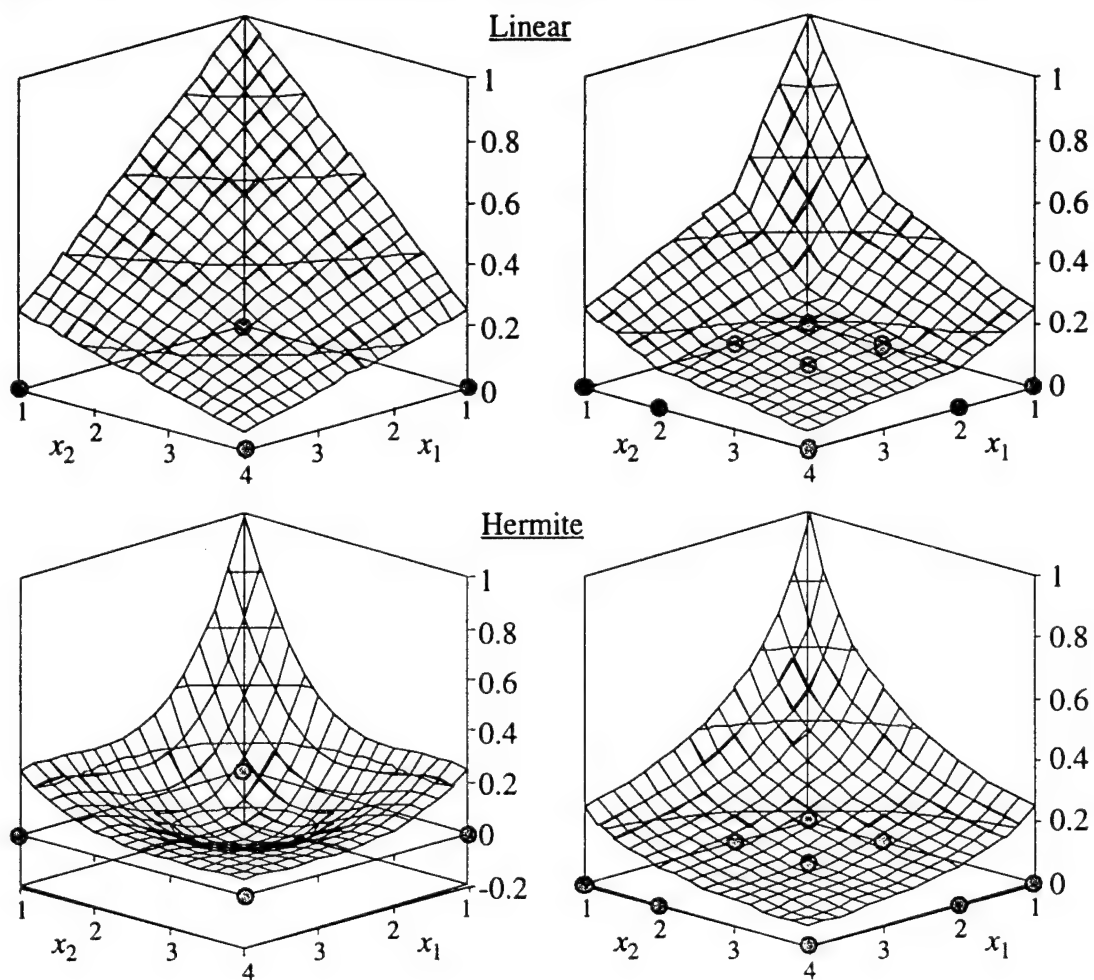


Figure 5E6. 2-D First-Order Hermite Interpolation of the Function $F(\mathbf{x}) = (x_1 x_2)^{-1}$



F. CONVEXITY OF THE FIRST-ORDER METHOD

Frequently, the highest costs (and highest marginal costs) are associated with extreme states of a system and cost-to-go functions are convex. When a cost-to-go is convex, it is likely that the total cost function (i.e., the sum of the current cost and the cost-to-go) is a convex function of control decisions. Under these conditions, search methods can find the globally optimal solution and will not find a “local” minimum that yields incorrect control decisions (however, it is possible to find correct control decisions even if a DDP cost-to-go function is not convex).

It is important that interpolation preserve the convexity of a cost-to-go function. Unfortunately, the Hermite interpolation proposed in the previous section does not always preserve convexity. The interpolation is convex over a subdomain of x (i.e., for all $0 \leq \xi_j \leq 1, j = 1, \dots, n$) only when the Hessian is positive semidefinite [Ecker and Kupferschid, 1991, p. 271]. A function is positive semidefinite if and only if all principle minors (i.e., the determinates of the square submatrices whose (1,1) elements are the (1,1) element of the Hessian) are nonnegative [Ecker and Kupferschid, 1991, p. 298].

1. Convexity of One-Dimensional Interpolation

For the 1-D case, the Hessian is a one by one matrix and there is only one principle minor. This principle minor is the second derivative

$$\frac{d^2 \hat{F}}{dx^2} = \frac{6}{\Delta x} \left[(1-2\xi) \frac{F_{i+1}-F_i}{\Delta x} + (\xi-\frac{2}{3})G_i + (\xi-\frac{1}{3})G_{i+1} \right] \quad (5F1)$$

where ξ is the local coordinate transformation of the single state variable x . The principle minor is a linear function of ξ . Thus, the principle minor is nonnegative over the domain $0 \leq \xi \leq 1$ if it is nonnegative at the bounding values at $\xi = \{0,1\}$. Bounding values are non-negative if and only if

$$\frac{2}{3}G_i + \frac{1}{3}G_{i+1} \leq \frac{F_{i+1}-F_i}{\Delta x} \leq \frac{1}{3}G_i + \frac{2}{3}G_{i+1} \quad (5F2)$$

The cost-to-go approximation is convex if these constraints are satisfied for each pair of adjacent nodes.

Figure 5E1 illustrates a function approximation using two different discretizations of the state variable. Interpolation using the coarser discretization is not convex even though the true function is strictly convex. In this case, the second derivative is negative at the node $x = 4$, and the upper bound constraint of equation (5F2) is violated. On the

other hand, interpolation using the finer discretization is convex and the constraints of equation (5E2) are satisfied.

This illustration suggests a possible solution for non-convex function approximations. For the example considered, the constraints are satisfied as the discretization interval Δx decreases. In most cases, we can achieve a convex function approximation that satisfies the constraints of equation (5F2) by selecting a discretization interval Δx that is sufficiently small. For smooth functions (i.e., functions with continuous first derivatives), non-convex features become small or disappear as the values G_i , G_{i+1} , and $(F_{i+1}-F_i)/\Delta x$ converge. Even if the constraints of equation (5F2) are not satisfied, we need not continue to decrease Δx until the cost-to-go function approximation is strictly convex. We can tolerate some small concave features in a function approximation because the distribution of outcomes produced by stochastic inputs. Even if there are small concave features in a cost-to-go estimate, the distribution of outcomes causes an averaging process that can still produce a convex total-cost estimate.

Foufoula Georgiou [1991] also addresses the problem of interpolating a 1-D convex function with convex interpolants. Using exponential functions, she produces interpolants that are strictly convex. Unfortunately, the interpolants have unbounded second derivatives and require that multiple interpolants be used. This can prevent application to multidimensional interpolation: when different interpolants are required for different subdomains, the resulting approximation will be discontinuous between subdomains.

2. Convexity of Multi-Dimensional Interpolation

For multi-dimension approximating functionals, the Hessian is more complex and evaluation of the principle minors is more difficult. For example, the (1,1) element of the 2-D Hessian is

$$\frac{d^2 \hat{F}}{dx_1^2} = \frac{d^2 f_0}{dx_1^2} + \frac{d^2 f_1}{dx_1^2} + \frac{d^2 f_2}{dx_1^2}$$

where
$$\frac{d^2 f_0}{dx_1^2} = \frac{6}{\Delta x_1} (1-2\xi_1) [P_{(1)} \frac{F_{\gamma} - F_{\beta}}{\Delta x} + P'_{(1)} \frac{F_{\gamma} - F_{\beta}}{\Delta x}]$$

$$\frac{d^2 f_1}{dx_1^2} = \frac{-4+6\xi_1}{\Delta x_1} [P_{(1)} G_{\gamma,1} + P'_{(1)} G_{\beta,1}] + \frac{-2+6\xi_1}{\Delta x_1} [P_{(1)} G_{\gamma,1} + P'_{(1)} G_{\gamma,1}]$$

$$\frac{d^2 f_2}{dx_1^2} = \frac{6}{\Delta x_1} (1-2\xi_1)[Q_2(G_{\gamma,2}-G_{\gamma,2}) - Q'_2(G_{\gamma,2}-G_{\gamma,2})] \frac{\Delta x_2}{\Delta x_1}$$

The definitions for $P_{(1)}$, $P'_{(1)}$, Q_2 , and Q'_2 are as given previously. It is difficult to identify simple convexity constraints as in the 1-D case.

Instead of evaluating convexity constraints for the multi-dimension interpolation, we might guess that if the 1-D convexity constraints are satisfied for each dimension, then the multi-dimensional interpolation is convex. This would seem reasonable since the proposed multi-dimensional interpolating functional is the n -fold product of 1-D functions. However, application of the proposed interpolation shows that it is not convex. However, among the various other interpolation methods attempted (including lower-order methods and non-polynomial methods), the proposed n -fold third-order polynomial weighting functions perform best.

Problems with non-convexity (and with poor cost-to-go estimates) appear to be worse when the off-diagonal elements of the Hessian are significant. As a result, the next sections present higher-order interpolation methods that use estimates of some second derivatives to reduce non-convex features of cost-to-go approximations. We will observe the impact of off-diagonal elements on multi-reservoir solutions analyzed in the next chapter.

G. A HERMITE INTERPOLATION METHOD WITH CONTINUOUS SECOND DERIVATIVES

Higher-order interpolation methods can preserve higher-order derivatives or produce a higher degree of smoothness. Preserving higher-order derivatives (e.g., second derivatives) should improve accuracy, but requires higher-order weighting functions and additional effort to evaluate and store the derivatives. Producing a higher degree of smoothness should improve convergence of search routines and may improve convexity, but also requires higher-order weighting functions.

This section presents a second interpolation method that produces a higher degree of smoothness by ensuring continuity of second derivatives between subdomains. Ideally, we would like an interpolation that produces continuous first and second to improve the efficiency of Newton-based search methods. Also, an interpolation with continuous first and second derivatives may preserve convexity better than the first-order Hermite method.

Additional requirements on the weighting functions are required to ensure continuous second derivatives. As with cost-to-go values and first derivatives, second derivatives along shared boundaries must be calculated to produce the same values in adjacent hypercubes. As a result, nodes and geometry that are not shared cannot be used to estimate second derivatives along a boundary. Table 5G1 identifies requirements on weighting functions $\phi(\eta)$ and $\psi_j(\eta)$ that will produce continuous second derivatives on an irregular grid (if a grid is regular, the constraints on $\eta_k = 0$ are not needed).

Continuity of second derivatives is accomplished without calculating the value of second derivatives at nodes. Instead, the requirements of Table 5G1 force the second derivatives to zero at hypercube boundaries. This may result in less accurate cost-to-go estimates; however, this avoids the potentially difficult task of calculating and storing these second derivatives.

Table 5G1: Weighting-Function Requirements for Second-Derivative Continuity on an Irregular Grid

	2nd Derivatives
$\phi(\eta)$	0 where $\eta_k = 0$ any $k = 1, \dots, n$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$
$\psi_j(\eta)$	0 where $\eta_k = 0$ any $k = 1, \dots, n$, 0 where $\eta_k = 1$ any $k = 1, \dots, n$

1. The Weighting Functions

Evaluating a weighting function as an n -fold product, there are five or six constraints on each 1-D function. These are constraints on the weighing function, its first derivative, and its second derivative at the two bounds $\eta_k = 0$ and $\eta_k = 1$, where k identifies the current dimension. As a result, each 1-D function is a fifth-order polynomial and the weighting functions are given by

$$\phi(\eta) = P = \prod_{k=1}^n (1+3\eta_k+6\eta_k^2)(1-\eta_k)^3$$

$$\psi_j(\eta) = Q_j P_{(j)} (d\eta_j/dx_j)^{-1}$$

where

$$Q_j = (1+3\eta_j)\eta_j(1-\eta_j)^3$$

$$P_{(j)} = \prod_{k=1, k \neq j}^n (1+3\eta_k+6\eta_k^2)(1-\eta_k)^3$$

If the grid is regular in some or all dimensions, we could define lower-order polynomials that do not require weights to vanish when $\eta_k = 0$ for some dimensions $k = 1, \dots, n$.

The 1st derivatives of the above polynomial weighting functions are

$$\begin{aligned}\frac{d\phi}{dx_s} &= (-30)\eta_s^2(1-\eta_s)^2P_{(s)} \frac{d\eta_s}{dx_s} \\ \frac{d\psi_j}{dx_s} &= (1+5\eta_s)(1-3\eta_s)(1-\eta_s)^2P_{(s)} & s=j \\ \frac{d\psi_j}{dx_s} &= (-30)\eta_s^2(1-\eta_s)^2Q_jP_{(s,j)}(d\eta_j/dx_j)^{-1} \frac{d\eta_s}{dx_s} & s \neq j\end{aligned}$$

and the 2nd derivatives are

$$\begin{aligned}\frac{d^2\phi}{dx_s^2} &= (-60)(1-2\eta_s)\eta_s(1-\eta_s)P_{(s)} \left(\frac{d\eta_s}{dx_s}\right)^2 \\ \frac{d^2\psi_j}{dx_s^2} &= (-12)(3-5\eta_s)\eta_s(1-\eta_s)P_{(s)} \frac{d\eta_s}{dx_s} & s=j \\ \frac{d^2\psi_j}{dx_s^2} &= (d\eta_j/dx_j)^{-1} \left(\frac{d\eta_s}{dx_s}\right)^2 (-60)(1-2\eta_s)\eta_s(1-\eta_s)Q_jP_{(s,j)} & s \neq j\end{aligned}$$

It can be easily verified that these weighting functions possess the properties of Table 5C4 and Table 5G1.

2. The One-Dimension Approximating Functional

The 1-D approximating functional and derivatives are

$$\begin{aligned}\hat{F}^{(i)}(x) &= PF_i + P'F_{i+1} + QG_i\Delta x - Q'G_{i+1}\Delta x \\ \frac{d\hat{F}^{(i)}}{dx} &= 30\xi^2(1-\xi)^2 \frac{F_{i+1}-F_i}{\Delta x} + (1+5\xi)(1-3\xi)(1-\xi)^2G_i - (6-5\xi)(2-3\xi)\xi^2G_{i+1} \\ \frac{d^2\hat{F}^{(i)}}{dx^2} &= (60)\xi(1-\xi)[(1-2\xi)\frac{F_{i+1}-F_i}{\Delta x} + (\xi-\frac{3}{5})G_i + (\xi-\frac{2}{5})G_{i+1}]/\Delta x\end{aligned}$$

Consistent with the definitions above,

$$\text{when } \eta = \xi \quad P = (1+3\xi+6\xi^2)(1-\xi)^3$$

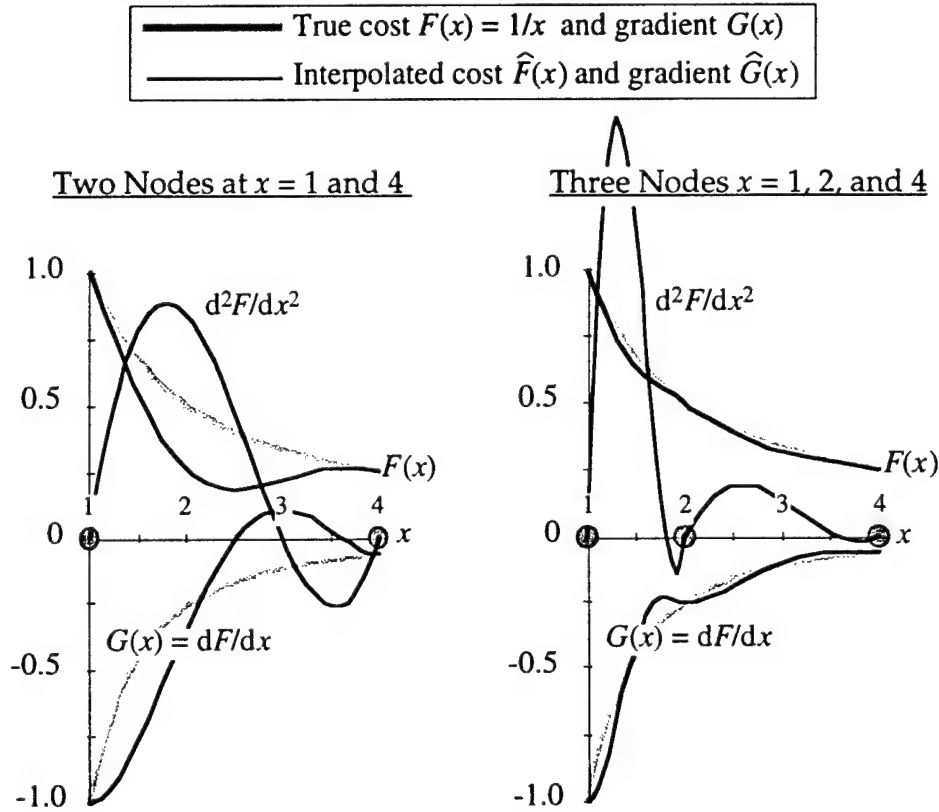
$$Q = (1+3\xi)\xi(1-\xi)^3$$

$$\text{when } \eta = 1-\xi \quad P' = (3-15\xi+6\xi^2)\xi^3$$

$$Q' = (4-3\xi)(1-\xi)\xi^3$$

Figure 5G1 illustrates this 1-D approximating functional. Comparing this figure with Figure 5E1, we see that this interpolation is less accurate and poorly preserves convexity of the true cost-to-go. Also, the curvature of the approximating functional increases with smaller discretization interval Δx .

Figure 5G1. 1-D Hermite Interpolation with Continuous Second Derivatives of the Function $F(x) = x^{-1}$



3. The Two-Dimension Approximating Functional

The 2-D approximating functional is

$$\hat{F}(\mathbf{x}) = f_0 + f_1 + f_2$$

where $f_0 = P_{(2)}[P_{(1)}F_{\gamma_1} + P'_{(1)}F_{\gamma_2}] + P'_{(2)}[P_{(1)}F_{\gamma_2} + P'_{(1)}F_{\gamma_1}]$

$$f_1 = Q_1[P_{(1)}G_{\gamma_{1,1}} + P'_{(1)}G_{\gamma_{2,1}}]\Delta x_1 - Q'_1[P_{(1)}G_{\gamma_{2,1}} + P'_{(1)}G_{\gamma_{1,1}}]\Delta x_1$$

$$f_2 = P_{(2)}[Q_2G_{\gamma_{1,2}} - Q'_2G_{\gamma_{2,2}}]\Delta x_2 - P'_{(2)}[Q_2G_{\gamma_{2,2}} + Q'_2G_{\gamma_{1,2}}]\Delta x_2$$

The 2-D interpolated 1st derivative with respect to x_1 is

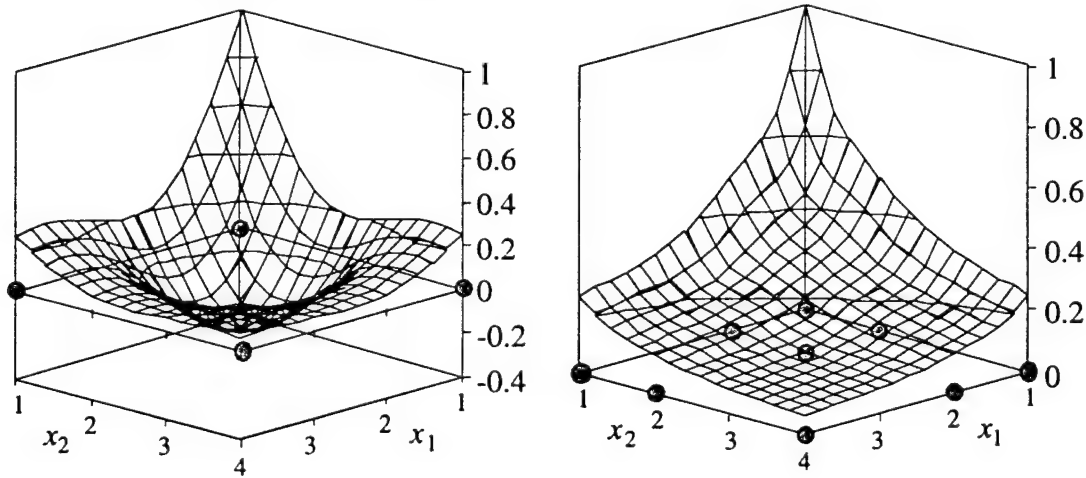
$$\frac{d\hat{F}}{dx_1} = \frac{df_0}{dx_1} + \frac{df_1}{dx_1} + \frac{df_2}{dx_1}$$

where

$$\begin{aligned} \frac{df_0}{dx_1} &= (30)\xi_1^2(1-\xi_1)^2[P_{(1)}\frac{F_{\gamma_2}-F_{\gamma_1}}{\Delta x_1} + P'_{(1)}\frac{F_{\gamma_2}-F_{\gamma_3}}{\Delta x_1}] \\ \frac{df_1}{dx_1} &= (1+5\xi_1)(1-3\xi_1)(1-\xi_1)^2[P_{(1)}G_{\gamma_1,1} + P'_{(1)}G_{\gamma_3,1}] \\ &\quad - (6-5\xi_1)(2-3\xi_1)\xi_1^2[P_{(1)}G_{\gamma_2,1} + P'_{(1)}G_{\gamma_4,1}] \\ \frac{df_2}{dx_1} &= (30)\xi_1^2(1-\xi_1)^2[Q_2(G_{\gamma_2,2}-G_{\gamma_1,2}) - Q'_2(G_{\gamma_4,2}-G_{\gamma_3,2})]\Delta x_2/\Delta x_1 \end{aligned}$$

The first derivative for $d\hat{F}/dx_2$ parallels that for $d\hat{F}/dx_1$. Figure 5G2 illustrates this 2-D approximating functional. Comparing this figure with Figure 5E6, we again see that this interpolation is less accurate and poorly preserves convexity of the true cost-to-go.

Figure 5G2. 2-D Hermite Interpolation with Continuous Second Derivatives of the Function $F(\mathbf{x}) = (x_1x_2)^{-1}$



4. Accuracy and Convexity of Hermite Interpolation with Continuous Second Derivatives

We see from the 1-D and 2-D examples that higher-order weighting functions may not produce interpolations as good as the simpler first-order Hermite method. To achieve continuous second derivatives, we produce an interpolation that is less accurate and has more difficulty preserving the convexity of a true cost-to-go function. In large part, this results from error induced by forcing second derivatives to zero at boundaries

between subdomains; however, this also demonstrates that interpolation functionals oscillate more severely when we use higher-order polynomials to produce weights.

As we observed, the interpolating functional has more difficulty preserving the convexity of a true cost-to-go function. For the 1-D approximating functional, the second derivative is nonnegative if

$$\frac{3}{5}G_i + \frac{2}{5}G_{i+1} \leq \frac{F_{i+1}-F_i}{\Delta x} \leq \frac{2}{5}G_i + \frac{3}{5}G_{i+1}$$

As we expect, the bounds of this equation are more restrictive than the bounds of equation (5F2) for the first-order Hermite interpolation. For the 2-D approximating functional, the (1,1) element of the Hessian is

$$\frac{d^2 \hat{F}}{dx_1^2} = \frac{d^2 f_0}{dx_1^2} + \frac{d^2 f_1}{dx_1^2} + \frac{d^2 f_2}{dx_1^2}$$

where
$$\frac{d^2 f_0}{dx_1^2} = \frac{60(1-2\xi_1)}{\Delta x_1} \xi_1(1-\xi_1) \left[P_{(1)} \frac{F_{\gamma_2}-F_{\gamma_1}}{\Delta x_1} + P'_{(1)} \frac{F_{\gamma_3}-F_{\gamma_2}}{\Delta x_1} \right]$$

$$\frac{d^2 f_1}{dx_1^2} = \frac{-12}{\Delta x_1} \xi_1(1-\xi_1) [(3-5\xi_1)(P_{(1)}G_{\gamma_{1,1}} + P'_{(1)}G_{\gamma_{3,1}}) + (2-5\xi_1)(P_{(1)}G_{\gamma_{2,1}} + P'_{(1)}G_{\gamma_{4,1}})]$$

$$\frac{d^2 f_2}{dx_1^2} = \frac{60}{\Delta x_1} \xi_1(1-\xi_1)(1-2\xi_1) [Q_2(G_{\gamma_{2,2}}-G_{\gamma_{1,2}}) - Q'_2(G_{\gamma_{4,2}}-G_{\gamma_{3,2}})] \frac{\Delta x_2}{\Delta x_1}$$

Again, it is difficult to identify simple convexity constraints as in the 1-D case.

Instead of forcing second derivatives to zero at boundaries, we could estimate second derivatives. From quasi-Newton search methods, we can extract estimates both of gradients and of the Hessian; however, such an estimate of the Hessian may not be sufficiently accurate. As an alternative, we can estimate the Hessian by finite differences using values of the gradient at adjacent nodes. Though this estimate is rough, it should be a significant improvement over forcing derivatives to zero as done in this section. Though computational effort and memory both increase, the increase in interpolation accuracy should prevent an overall increase in effort and memory.

H. A SECOND-ORDER HERMITE INTERPOLATION METHOD

This section presents a third interpolation method that uses off-diagonal elements of the Hessian to produce more accurate estimates. This additional derivative information improves the convexity of interpolating functionals. What is more important, this method is able to use these derivatives without increasing the order of polynomials used by the weighting functions. However, this method does not produce continuous second derivatives.

The off-diagonal elements of the Hessian are the elements $d^2F/dx_j dx_k$ where $j \neq k$. The interpolating functional is given by the weighted sum

$$\hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_n} \{ \phi_\gamma(\mathbf{x}) F_\gamma + \sum_{j=1}^n \{ \psi_{\gamma,j}(\mathbf{x}) G_{\gamma,j} + \sum_{k=j}^n \{ \chi_{\gamma,j,k}(\mathbf{x}) H_{\gamma,j,k} \} \} \} \quad (5H1)$$

Table 5H identifies requirements on the second-derivative weights $\chi_{j,k}(\mathbf{x})$ required along with the requirements of Table 5C4 to produce continuous and smooth interpolations. Diagonal elements of the Hessian (i.e., d^2F/dx_j^2) are not used as this requires use of higher-order polynomials as in the last section.

A great advantage of this second-order method is that the second-derivative weighting function $\chi_{j,k}(\mathbf{x})$ is constructed from the same 1-D polynomials used by the first-order Hermite method. Also, the weighting functions $\phi(\mathbf{x})$ and $\psi_j(\mathbf{x})$ are the same as those used for the first-order Hermite method.

Table 5H1. Second-Derivative Weighting-Function Requirements for Continuity and Smoothness on an Irregular Grid

	Zero'th-Order Value	First derivative in dimension x_s	Second derivative in dimensions x_r and x_s , $r \neq s$
$\chi_{j,k}(\eta)$	0 where $\eta_j = 0$, 0 where $\eta_k = 0$, 0 where $\eta_q = 1$ any q	0 where $\eta_j = 0$, $s \neq j$, 0 where $\eta_k = 0$, $s \neq k$, 0 where $\eta_s = 0$, $s \notin \{k, j\}$ 0 where $\eta_q = 1$ any q	1 where $\eta = 0$, $\{r, s\} = \{j, k\}$, 0 where $\eta_j = 0$, $r, s \neq j$, 0 where $\eta_k = 0$, $r, s \neq k$, 0 where $\eta_r = 0$, $r \notin \{k, j\}$ 0 where $\eta_s = 0$, $s \notin \{k, j\}$ 0 where $\eta_q = 1$ any q

1. Shorthand Notation

To present the second-order method, we will use shorthand notation for the polynomials used to construct the weighting functions. The following third-order polynomials are the same 1-D polynomials used by the first-order weighting functions used to satisfy the requirements of Table 5C4:

$$\alpha_j = (1+2\eta_j)(1-\eta_j)^2$$

$$\beta_j = \eta_j(1-\eta_j)^2 (d\eta_j/dx_j)^{-1}$$

In addition, the following second-order polynomials are derivatives of the weighting function polynomials

$$\omega_j = \frac{d\alpha_j}{dx_j} = -6\eta_j(1-\eta_j) \frac{d\eta_j}{dx_j}$$

$$\delta_j = \frac{d\beta_j}{dx_j} = (1-3\eta_j)(1-\eta_j)$$

Using this notation, the first-order weighting functions are

$$\phi(\eta) = P = \prod_{k=1}^n \alpha_k$$

$$\psi_j(\eta) = \beta_j P_{(j)}$$

where

$$P_{(j)} = \prod_{k=1, k \neq j}^n \alpha_k$$

The first derivatives are

$$\frac{d\phi}{dx_s} = \omega_s P_{(s)}$$

$$\frac{d\psi_j}{dx_s} = \delta_s P_{(s)} \quad s = j$$

$$\frac{d\psi_j}{dx_s} = \omega_s \beta_j P_{(s,j)} \quad s \neq j$$

And the second derivatives $d^2/dx_r dx_s$, $r \neq s$, are

$$\frac{d^2\phi}{dx_r dx_s} = \omega_r \omega_s P_{(r,s)}$$

$$\frac{d^2\psi_j}{dx_r dx_s} = \omega_r \delta_j P_{(r,j)} \quad s=j$$

$$\frac{d^2\psi_j}{dx_r dx_s} = \omega_s \delta_j P_{(s,j)} \quad r=j$$

$$\frac{d^2\psi_j}{dx_r dx_s} = \omega_r \omega_s \beta_j P_{(r,s,j)} \quad r,s \neq j$$

2. Weighting Function for the Second Derivatives

As mentioned, the second-derivative weighting function $\chi_{j,k}(\mathbf{x})$ is constructed from the same 1-D polynomials used by the first-order Hermite method. This function is

$$\chi_{k,j}(\eta) = \beta_k \beta_j P_{(k,j)}$$

Its first derivatives are

$$\frac{d\chi_{k,j}}{dx_s} = \beta_k \delta_j P_{(k,j)} \quad s=j$$

$$\frac{d\chi_{k,j}}{dx_s} = \delta_k \beta_j P_{(k,j)} \quad s=k$$

$$\frac{d\chi_{k,j}}{dx_s} = \omega_j \beta_k \beta_j P_{(s,k,j)} \quad s \notin \{k,j\}$$

And its second derivatives $d^2/dx_r dx_s$, $r \neq s$, are

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \delta_k \delta_j P_{(k,j)} \quad \{r,s\} = \{k,j\}$$

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \omega_r \beta_k \delta_j P_{(r,k,j)} \quad r \neq k, s=j$$

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \omega_r \delta_k \beta_j P_{(r,k,j)} \quad r \neq j, s=k$$

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \omega_s \beta_k \delta_j P_{(s,k,j)} \quad r=j, s \neq k$$

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \omega_s \delta_k \beta_j P_{(s,k,j)} \quad r=k, s \neq j$$

$$\frac{d^2\chi_{k,j}}{dx_r dx_s} = \omega_r \omega_s \beta_k \beta_j P_{(r,s,k,j)} \quad r,s \notin \{k,j\}$$

To verify that this weighting function and its derivatives satisfy the requirements of Table 5H, notice that all polynomials vanish when $\eta = 1$ and that β and ω vanish when $\eta = 0$. This is appropriate since β and ω depend on the discretization interval Δx_k .

I. COMPUTATIONAL EFFICIENCY OF METHODS

The effort required to solve a DDP problem is proportional to a number of discrete states that increases exponentially with the number of state variables as described by equation (4C1). For each discrete state, we solve a smaller optimization problem that identifies the best control decisions for this initial state. Most execution time is spent in the search routine used to find optimal decisions for each discrete state. As a result, we can also reduce the total execution time for a DDP problem by reducing this search time.

The first way to reduce search time is to use an efficient search routine. The amount of time required by each search depends on the number of times the total cost function of equation (4A5) is evaluated. More efficient search routines (such as Newton-based routines) can find the optimal decisions with fewer the evaluations, and can significantly reduce the execution time.

The second way to reduce search time is to reduce the amount of time required to evaluate the total cost function. For DDP problems with multiple state variables, the amount of time required to evaluate the total cost function depends primarily on the time required to interpolate the cost-to-go. Usually, the effort required to evaluate the interpolating functional will be large compared to the effort required to evaluate the current cost function or the state-transition function. Even in the case of crude nearest neighbor interpolation, a significant time amount of time may be spent searching for the correct subdomain.

To evaluate the new Hermite interpolation methods, it is useful to identify the hypothetical effort required to interpolate using the various methods. In particular, we are interested in seeing how the effort grows with the dimension of the interpolation (i.e., the number of state variables). One approach is to count the number of "flops," or floating-point operations required to perform each interpolation [Johnson *et al.*, 1993]. A flop is a floating point multiplication and an addition (with most effort due to the multiplication).

To evaluate the cost-to-go at a state x requires effort to evaluate the weighting functions and additional effort to apply the weights to node values. Let us first consider the effort required to apply weights to nodes. In the case of a local approximation that uses only immediately adjacent nodes (i.e., corner nodes of the surrounding hypercube),

there are 2^n nodes. Thus, for multilinear interpolation, 2^n flops are required to apply the weights to the cost-to-go value at each node and sum the result. For the first-order Hermite interpolation method, $2^n(n+1)$ flops are required to apply the weight to the cost-to-go value and derivatives at each node. For the second-order Hermite interpolation method, $2^{n-1}(n^2+n+2)$ flops are required for the additional $(n^2-n)/2$ second derivatives. This does not include the diagonal elements of the Hessian and assumes that the Hessian is symmetric (i.e., the (j,k) element equals the (k,j) element). In contrast, spline methods require a more extensive set of nodes but do not use derivatives to estimate the cost-to-go. In this case, there may be on the order of 4^n flops [Johnson *et al.*, 1993]. Clearly, the effort to attain higher-order accuracy using Hermite interpolation or splines can result in an exponential increase in interpolation effort.

Now let us consider the effort to evaluate the weighting functions. The 1-D functions $\alpha(\eta_j)$, $\beta(\eta_j)$, $\alpha(\eta_j)$, and $\delta(\eta_j)$, $j=1,\dots,n$, are used numerous times in different weighting functions. Careful programming can significantly reduce the effort to evaluate the weighting functions by reusing in different weights the evaluation of the 1-D functions and their products. In contrast, it is not as easy to reduce the effort of the original Hermite interpolation [Kitanidis, 1986] because the weights cannot be decomposed into as simple a set of functions. Table 5I1 identifies the effort to interpolate the cost-to-go value using the linear and Hermite interpolation methods. The total effort is the sum of the effort to evaluate the 1-D functions, the effort to evaluate the weights using these functions, and the effort to apply these weights to the nodes (there are also n divide operations required to evaluate $(\Delta x)^{-1}$, but this effort grows only linearly with the number state variables). With careful programming, the effort to evaluate the weights is only twice the effort to apply the weights.

Table 5I2 identifies the growth in effort with dimension for multilinear interpolation, the first-order Hermite method, and the second-order Hermite method. These are the interpolation methods that have been included in the DDP code presented in an appendix to this thesis, and the effort is evaluated by counting the flops in the code. The effort to interpolate the cost-to-go F is identified for each method to provide an equal basis for comparison. Implementation of the quasi-Newton search is more efficient if we calculate the gradient of the cost to go analytically, and actual implementation in the GDP code interpolates both F and G .

The next chapter will apply these three methods to a range of multi-reservoir problems to verify the interpolation effort anticipated here and to observe the accuracy of each method. Table 5I2 indicates that the first-order Hermite method requires approximately an n -fold increase in effort to evaluate the cost-to-go. Actual

implementation of the first-order Hermite method interpolates both the cost-to-go and its gradient, and this should increase the effort approximately by a factor of n^2 .

Equivalently, the effort of the second-order Hermite method is approximately a factor $n/2$ greater than the effort for the first-order Hermite method.

Table 5I1. Distribution of Effort for One Interpolation of the Cost-To-Go (in Flops)

Method	Evaluate 1-D polynomials	Evaluate weights	Apply weights	Total effort
Linear	n	$\sim 2^{n+1}$	2^n	$\sim 3 \cdot 2^n$
First-Order Hermite	$14n$	$\sim 2^{n+1}n$	$2^{n(n+1)}$	$\sim 3 \cdot 2^n n$
Continuous second derivatives	$20n$	$\sim 2^{n+1}n$	$2^{n(n+1)}$	$\sim 3 \cdot 2^n n$
Second-Order Hermite	$14n$	$\sim 2^n n^2$	$2^{n-1}(n^2+n+2)$	$\sim 3 \cdot 2^{n-1} n^2$

Table 5I2: Total Flops For Each Evaluation of the Cost-to-Go (per Code)

Number of dimensions	Linear	First-Order Hermite		Second-Order Hermite	
	(F)	(F)	(F and G)	(F)	(F and G)
1	3	13	22	n.a.	n.a.
2	10	36	94	42	116
3	23	97	328	147	526
4	48	234	1014	452	2044
5	97	547	2932	1309	7162
...
n	$\sim 3 \cdot 2^n$	$\sim 3 \cdot 2^n n$	$\sim 3 \cdot 2^n n^2$	$\sim 3 \cdot 2^{n-1} n^2$	$\sim 3 \cdot 2^{n-1} n^3$

CHAPTER 6.

ANALYSIS OF GRADIENT DYNAMIC PROGRAMMING

In this chapter, we evaluate the benefit of the first-order and second-order Hermite interpolation methods of Chapter Five. We will apply these methods to solve reservoir control problems with a range of complexity. As a basis for comparison, we will also apply the multilinear interpolation method of Chapter Four to solve many of the same problems. All three methods are applied consistently and are incorporated in the computer code included in Appendix B.

Each method is compared by evaluating the trade-off between solution accuracy and time to execute the code. This is accomplished by applying each method to reservoir control models using progressively finer state-space grids to approximate the cost-to-go functions. These finer grids yield solutions of increasing accuracy but also require increasing time to execute the code. As we will see, the solutions validate the expected performance discussed in Chapters Three and Five and support the accurate application of GDP to stochastic dynamic control problems with six or more state variables.

The following analysis closely parallels the analysis of *Johnson et al.* [1993] used to evaluate the benefit of spline methods. *Johnson et al.* applied spline and linear interpolation methods to solve reservoir control problems with between two and five reservoirs. They also apply the original Hermite interpolation method [*Foufoula Georgiou and Kitanidis*, 1988] to a four-reservoir problem and demonstrate that the original Hermite interpolation performs about as well as spline methods. The applications of spline and Hermite methods demonstrate that higher-order interpolation allows us to solve stochastic DDP problems of higher dimension than possible with traditional DDP methods.

A. THE SERIES OF MULTI-RESERVOIR TEST PROBLEMS

This section presents the series of increasingly complex test problems used to test DDP using the linear and Hermite interpolation methods. First, we will consider a traditional four-reservoir model that has been used by previous authors to test their systems analysis methods. Second, we will consider a series of modifications to create models with between one and seven reservoirs, extrapolating on the examples of *Johnson et al.* [1993].

1. The Four-Reservoir Test Problem

Since its introduction by *Larson* [1968], the four-reservoir model has been used by many authors to test stochastic control methods [*Chow et al.*, 1975; *Foufoula Georgiou and Kitanidis*, 1988; *Heidari et al.*, 1971; *Kitanidis and Andricevic*, 1989; *Murray and Yakowitz*, 1979; *Saad et al.*, 1992; *Sobel*, 1989; *Yakowitz*, 1982]. Of particular interest, *Johnson et al.* [1993] used the four-reservoir model to assess the performance of spline DDP methods and *Foufoula-Georgiou and Kitanidis* [1988] used the model to test the original GDP algorithm.

In applications of this thesis (and in the applications of *Johnson et al.* and *Foufoula-Georgiou and Kitanidis*), the four-reservoir model is fed by two independent stochastic streamflows. The four reservoirs are distributed above and below the confluence of the two streams (Figure 6A1). Upstream reservoir 1 receives flow of the first stream. Reservoirs 2 and 3 receive flow of the second stream. Downstream reservoir 4 receives water released from reservoirs 1 and 3. Streamflows are uncorrelated (both with each other and in time), and the state of the system is simply described by storage levels in each of the four reservoirs. A solution identifies the release policy for each reservoir (as a function of storage levels) and the expected cost for any initial set of storage levels. The optimal solution minimizes the expected cost of operations in each of three stages. The cost of operation is a quadratic penalty of deviations from desired releases.

The mathematical model of the four-reservoir system follows the conventions established in Chapter Two. The four state variables $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$ represent storage in the four numbered reservoirs of Figure 6A1. The four decision variables $\mathbf{u} = [u_1, u_2, u_3, u_4]^T$ represent releases from each reservoir. Two stochastic variables $\mathbf{w} = [w_1, w_2]^T$ represent the uncorrelated streamflows into reservoirs #1 and #2. These

streamflows are normally distributed with mean values of $\mu = [2,4]^T$ and standard deviation $\sigma = [0.5,0.75]^T$ and are represented by discrete flows $w_1 \in \{1.5,2.5\}$ and $w_2 \in \{3.25,4.75\}$ weighted equally. For each of the three stages, the state transition function $y = T_t(x,u,w)$ is linear and can be expressed in matrix form as

$$T(u,x,w) = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 \end{bmatrix} u + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} w$$

Releases cannot be negative ($u \geq 0$) and reservoir levels must remain between $[0,12]$ ($0 \leq u \leq 12$).

The cost function in each of three stages is a quadratic function of deviations from desired releases. The desired releases in each stage are $u = 1$ for all reservoirs. The cost in each of the three stages $t = \{0, 1, 2\}$ is

$$C_t(u) = \sum_{j=1}^4 \{ a_j (u_{(t)j} - 1)^2 \}$$

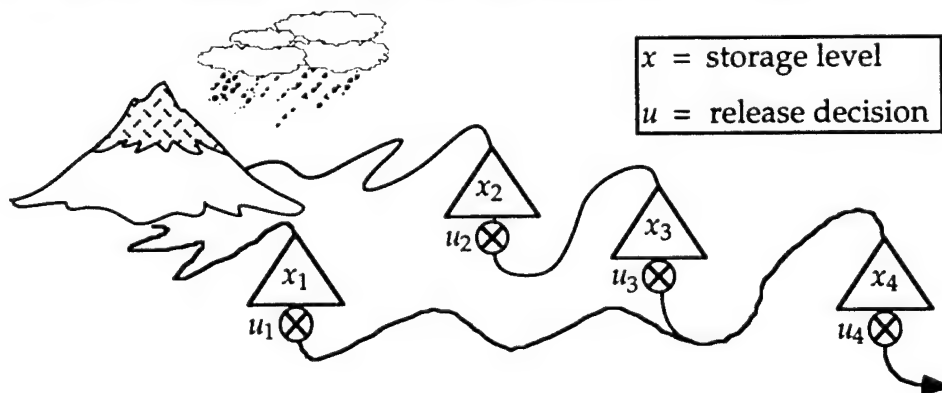
where $a = [1.1, 1.2, 1.0, 1.3]^T$. The cost-to-go at the time horizon (i.e., the cost for the terminal state) is given by the function

$$F_{t_3}(x) = \sum_{j=1}^4 \{ (x_j - b_j)^2 \}$$

where $b = [5.0, 5.0, 5.0, 7.0]^T$. The optimal solution minimizes the expected value of these costs for the entire operating horizon:

$$F_1(x) = \min_{U_{(t_1)}, U_{(t_2)}, U_{(t_3)}} \{ E_{w_{(t_1)}, w_{(t_2)}, w_{(t_3)}} \{ \sum_{t=t_1}^{t_3} \{ C_t(u) \} + F_{t_3}(x) \} \}$$

Figure 6A1. Illustration of the Four-Reservoir Control Problem



2. Formulation of the Multi-Reservoir Test Problems

Modifications of the four-reservoir problem are used to create a series of multi-reservoir models with varying complexity. Models with one to five reservoirs are those used by *Johnson et al.* [1993]. Models with six and seven reservoirs are simple extrapolations of the pre-existing models (Figure 6A2).

The current-cost functions used by the seven reservoir-system models are

$$C_t[\mathbf{u}] = \sum_{j=1}^n \{ a_j(u_j-1)^2 \}, \quad n = 1, 3, 4, 5, 6, 7$$

$$C_t[\mathbf{u}] = (u_1+u_2-2)^2, \quad n = 2$$

Note that the current-cost function used with the two-reservoir model is different from that used with the other six models. As we will see later, this difference causes the relationship between computational effort and accuracy to deviate from the trend established by the other six problems. Consistent with the four-reservoir problem, the final cost-to-go associated with the terminal state of each system is

$$F_3[\mathbf{x}] = \sum_{j=1}^n \{ (x_j-b_j)^2 \}$$

Constraints are $0 \leq \mathbf{u}$, and $0 \leq \mathbf{x} \leq 12$. The coefficients **a** and **b** for each of the seven models are given in Table 6A1.

Figure 6A2. Multi-Reservoir Systems

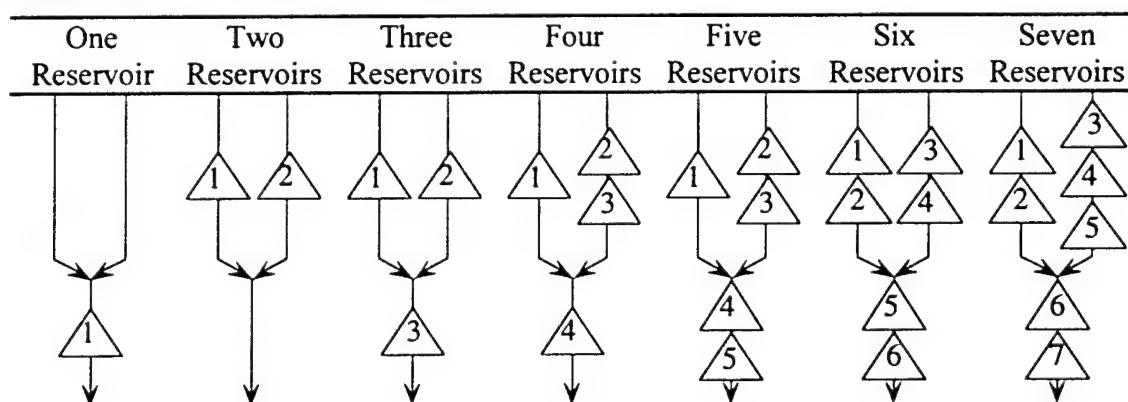


Table 6A1. Definition of Multi-Reservoir Problems

Case	Description	Parameters
$n = 1$	Reservoir below confluence of streams	$a = 1.0, b = 7$
$n = 2$	Reservoir on each stream above confluence	$\mathbf{a} = [1.0, 1.0]^T$ $\mathbf{b} = [5, 7]^T$
$n = 3$	Reservoir on each stream above confluence and reservoir below confluence	$\mathbf{a} = [1.1, 1.2, 1.3]^T$ $\mathbf{b} = [5, 5, 7]^T$
$n = 4$	Same as ($n=3$) with two reservoirs on stream 2	$\mathbf{a} = [1.1, 1.2, 1.0, 1.3]^T$ $\mathbf{b} = [5, 5, 5, 7]^T$
$n = 5$	Same as ($n=4$) with two reservoirs below confluence	$\mathbf{a} = [1.1, 1.2, 1.0, 1.3, 1.1]^T$ $\mathbf{b} = [5, 5, 5, 7, 7]^T$
$n = 6$	Same as ($n=5$) with two reservoirs on stream 1	$\mathbf{a} = [1.1, 1.2, 1.0, 1.3, 1.1, 1.0]^T$ $\mathbf{b} = [5, 5, 5, 7, 7, 7]^T$
$n = 7$	Same as ($n=6$) with three reservoirs on stream 2	$\mathbf{a} = [1.1, 1.2, 1.0, 1.3, 1.1, 1.0, 1.0]^T$ $\mathbf{b} = [5, 5, 5, 7, 7, 7, 7]^T$

B. COMPUTATIONAL EFFORT TO SOLVE THE SERIES OF MULTI-RESERVOIR TEST PROBLEMS

This section presents the computational effort required to solve the seven multi-reservoir test problems using the linear and Hermite interpolation methods. As we will see, the effort grows exponentially with the number of reservoirs (i.e., the number of state variables). In Chapter Four, we described this growth by the equation

$$J = Z \Lambda^n$$

where Z is the effort to identify optimal control decisions for one initial state of a one-stage subproblem. This effort grows exponentially with the number of state variables n and the discretization Λ .

However, this is not the complete story since Z also grows exponentially with the number of state variables. Chapter Four divided this effort into the product of Z_I , the time required to evaluate the total cost function, Z_S , the number of evaluations required to find the solution that minimizes the total cost, and Z_1 , a catch-all term for overhead and

other factors that influence the total effort J . Thus, we described the total effort by the equation

$$J = Z_I Z_L Z_S \Lambda^n$$

We will evaluate the computational effort to solve the seven multi-reservoir test problems as a function of these three factors.

This computational effort will depend on the search routines employed. When using Hermite interpolation, we will employ the quasi-Newton method of NPSOL [Gill *et al.*, 1986]. NPSOL iteratively searches for an optimal solution by using estimates of the cost-function gradient and Hessian to identify a search direction and by using a line-search routine to find new solutions. Constraints are incorporated by the Lagrange multipliers. When using linear interpolation, we will employ the downhill simplex method using the computer code of Press *et al.* [1992, pp. 402-6] in modified form (see Appendix). In brief, a simplex is a geometrical figure consisting of $n+1$ linearly independent vertices (e.g., in two dimensions, any triangle is a simplex). Vertices are added and dropped iteratively so that the simplex moves in an amoeba-like fashion to find the minimum. Constraints are incorporated by adding to the cost function a large penalty for constraint violations. A cost function can have discontinuous gradients since the method does not use gradients to identify search directions.

1. Standardization of Computational Time

The raw computational time of the solutions presented in this chapter are not directly comparable. These solutions were run on a half-dozen HP-9000 Series 700 workstations with different processor speeds. Also, computational time is in elapsed time, not in processor time; and multiple users of these machines have an impact on the raw computational time.

To standardize the computational time, a series of short runs was conducted to estimate the time consumed in each interpolation of the cost-to-go function. Because this time depends on the dimension of each problem and on the interpolation method employed, problems were solved for each combination of the different problems and the interpolation methods. These runs were conducted with no competition from other users on the fastest workstation (an HP-9000 755). The reported computational times are averages from four runs of each reservoir problem. The reported times are used to remove the impact of different computer speeds and loads on all other runs. Following the practice of Johnson *et al.* [1993], computational time and error are reported for the first stage of the three-stage time horizon (i.e., t_1).

The effort to evaluate the total cost equation (4A5) is the sum of efforts to evaluate the current-cost function, the transition function, and the future cost function. In general, the effort to evaluate the future cost function will predominate because of the relatively large effort to interpolate the cost-to-go. As a result, the standardized time consumed in each interpolation is a good estimate of Z_I , the time required to evaluate the total cost function.

LINEAR INTERPOLATION

Runs were conducted to solve problems with one to five reservoirs. Problems with six or seven reservoirs required considerable computational time, even for the crudest discretization ($\Delta = 2$). Problems with four or five reservoirs could be solved with coarse discretization; however, this does not imply an ability to solve these DDP problems using linear interpolation. As we will see later, the multilinear DDP solutions for the four- and five-reservoir problems contain significant error.

The time consumed in each interpolation of the cost-to-go is summarized in italics on the last line of Table 6B1. The runs were designed to be short, but with a sufficient number of interpolations that a stable estimate would result. As the dimension of the five problems increases, the discretization is coarser and the accuracy is lower. The total time per stage should not be used to infer the growth in total effort J with dimension, though the total time per node can be used to infer the growth in effort per node Z .

Table 6B1 preserves additional information that we can use to understand the growth in effort with dimension. For these runs and for runs presented later, the total time per stage is three to ten times the total time consumed in evaluating interpolants, regardless of dimension. A significant fraction of the total time is spent in program execution and calculations other than interpolation because multilinear interpolation is quicker and solver convergence is slower. For example, a significant fraction of the total time is consumed simply in identifying the current subdomain for interpolation. Large amounts of the total time are also consumed in other overhead activities, and it is likely that this overhead could be reduced by more efficient code. However, these overhead activities become a smaller fraction of the total time with increasing dimension. Also, multilinear DDP is used only as a benchmark against which we measure the performance of GDP. Thus, only limited work has been applied to reduce the overhead effort of the multilinear DDP code.

Table 6B1. Standard Computational Times per Stage of Linear Interpolation

# Reservoirs, n	One	Two	Three	Four	Five
Discretization, Δ	17	7	5	3	2
# nodes, Δ^n	17	49	125	81	32
# interpolations	3356	31254	119699	188827	191649
<u>Total time (seconds)</u>					
per stage	2.15	17.59	72.55	104.67	126.52
per node	.13	.36	.58	1.29	3.95
<u>Total time (seconds) consumed in</u>					
ID subdomain	.25	1.83	8.13	12.74	16.53
Eval. interpolants	.19	2.14	8.56	18.18	38.94
Overhead	1.71	13.62	55.86	73.75	71.05
<u>Time per interpolation (seconds) consumed in</u>					
ID subdomain	.00007	.00006	.00007	.00007	.00009
Eval. interpolants	.00006	.00006	.00007	.00010	.00020

FIRST-ORDER HERMITE INTERPOLATION

Runs were conducted to solve problems with one to seven reservoirs. The time consumed in each interpolation of the cost-to-go is summarized in italics on the last line of Table 6B2. The discretization of each problem is the same as used with the linear interpolation.

As expected, the time required for each interpolation is significantly greater for the first-order Hermite method than for the linear method. In Table 6B1, the time consumed in each interpolation is 60 to 200 microseconds (μsec); but, in Table 6B2, the time consumed in each interpolation is 50 to 7120 μsec .

However, what is lost in interpolation effort is more than recovered by the more rapid convergence of the quasi-Newton solver employed by GDP. Though the polytope solver employed by multilinear DDP is more robust and can handle discontinuous gradients, it is much slower. As a result, GDP requires fewer interpolations to find each solution, and the total interpolation time is consistently smaller by almost an order of magnitude. As a result, the total time per node in Table 6B1 is 0.13 to 3.95 seconds; but the total time per node in Table 6B2 is 0.05 seconds for the one-reservoir problem to 0.25 seconds for the five-reservoir problem.

The overhead activities for GDP rapidly become a small fraction of the total time per stage with increasing dimension. The total time per stage is over ten times the total time consumed in evaluating interpolants for low-dimension problems, but this decreases to less than 50% greater for the seven-reservoir problem. We can see that most of the effort required to solve higher dimension problems is consumed in evaluating interpolants

of the cost-to-go function. The effort required to identify the current subdomain for interpolation is insignificant except in low dimension problems (though not shown, this is also true even when $\Lambda > 2$ and identification of the current subdomain is not trivial).

Table 6B2. Standard Computational Times per Stage of First-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
Discretization, Λ	17	7	5	3	2	2	2
# nodes, Λ^n	17	49	125	81	32	64	128
# interpolations	522	3107	6985	5062	2095	4690	10179
<u>Total time (seconds)</u>							
per stage	.77	4.91	12.05	11.01	8.05	24.09	102.40
per node	.05	.10	.10	.14	.25	.38	.80
<u>Total time (seconds) consumed in</u>							
ID subdomain	.03	.22	.53	.42	.22	.65	1.90
Eval. interpolants	.03	.44	1.89	2.93	3.24	13.85	72.48
Overhead	.71	4.25	9.63	7.66	4.59	9.59	28.02
<u>Time per interpolation (seconds) consumed in</u>							
ID subdomain	.00006	.00007	.00008	.00008	.00011	.00014	.00019
<i>Eval. interpolants</i>	<i>.00005</i>	<i>.00014</i>	<i>.00027</i>	<i>.00058</i>	<i>.00155</i>	<i>.00295</i>	<i>.00712</i>

SECOND-ORDER HERMITE INTERPOLATION

As for the first-order Hermite interpolation method, runs were conducted to solve problems with one to seven reservoirs. The time consumed in each interpolation of the cost-to-go is summarized in italics on the last line of Table 6B3. The discretization of each problem is the same as used with linear and first-order Hermite interpolation.

As expected, the time required for each interpolation is the largest of all methods. In Table 6B3, the time consumed in each interpolation is 60 to 31,370 μsec . In addition, the additional time consumed in evaluating second derivatives by finite differences is from 110 to 15,610 μsec (though, with increasing dimension, this becomes a fraction of the time consumed by interpolation). As a result, the total time per node is greater than when using first-order Hermite interpolation. In Chapter 5, Section I, we predicted that the effort per interpolation should approach a factor n greater.

Table 6B3. Standard Computational Times per Stage of Second-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
Discretization, Δ	17	7	5	3	2	2	2
# nodes, Δ^n	17	49	125	81	32	64	128
# interpolations	522	3186	6488	4838	2041	4465	9694
<u>Total time (seconds)</u>							
per stage	1.75	6.07	14.61	17.44	16.32	85.74	477.08
per node	.10	.12	.12	.22	.51	1.34	3.73
<u>Total time (seconds) consumed in</u>							
ID subdomain	.03	.23	.51	.42	.21	.60	2.05
Eval. 2'nd deriv.	.06	.65	2.54	4.37	4.85	28.01	151.36
Eval. interpolants	.03	.54	2.77	5.50	7.19	48.41	304.09
Overhead	1.63	4.65	8.79	7.15	4.07	8.72	19.58
<u>Time per interpolation (seconds) consumed in</u>							
ID subdomain	.00006	.00007	.00008	.00009	.00010	.00013	.00021
Eval. 2'nd deriv.	.00011	.00020	.00039	.00090	.00238	.00627	.01561
Eval. interpolants	.00006	.00017	.00043	.00114	.00352	.01084	.03137

2. Growth in Interpolation Effort with State Dimension

At the end of Chapter Five, we considered the hypothetical interpolation effort of the multilinear and Hermite methods. The effort for all methods increases with dimension because the increasing number and complexity of weights that must be evaluated and applied. If the hypothetical interpolation effort provides a good approximation of the actual time consumed in each interpolation, then the hypothetical effort provides a good approximation of Z_I , the time required to evaluate the total cost function. They can also be used to identify inefficiency in algorithms and implementing code.

Table 5I1 summarizes the hypothetical effort (in flops) to interpolate the cost-to-go. Keeping the lower order terms to produce better estimates for the low-dimension problems, the hypothetical time per interpolation in multilinear DDP is

$$Z_I^{\text{lin}} \approx Z_0(n + 3 \cdot 2^n) \quad (6B1)$$

where Z_0 is a measure of computer processor speed. The hypothetical effort (in flops) for each interpolation method in GDP is

first-order Hermite: $14n + 2^n(3n+1)$

second-order Hermite: $14n + 2^{n-1}(3n^2+n+2)$

However, these do not provide good approximations of Z_I . As we will discuss later, each solution of the search routines is verified by a restart. In the case of GDP, the first call of the quasi-Newton method is used to search for the optimal control decisions and the total cost. The second call is then used to verify the previous solution and to estimate the cost-to-go gradient. Thus, the second call requires a factor $(n+1)$ greater effort to interpolate the cost-to-go and each of the n first derivatives. The average effort for each interpolation is the average of the effort on the first call and the effort on the second call. Thus, the hypothetical effort in GDP is

$$Z_I^{H1} \approx Z_0(14n + 2^n(3n+1) \frac{n+2}{2}) \quad (6B2)$$

$$Z_I^{H2} \approx Z_0(14n + 2^{n-1}(3n^2+n+2) \frac{n+2}{2}) \quad (6B3)$$

where Z_I^{H1} and Z_I^{H2} are the hypothetical times per interpolation using first-order and second-order Hermite methods.

Table 6B4 summarizes the observed and hypothetical effort for one to seven reservoirs using the linear and Hermite interpolation methods. The ratios of these provide an estimate of Z_0 , the time per flop. For each interpolation method, the estimate of Z_0 decreases with the dimension of the problem as overhead effort not included in equations (6B1) through (6B3) becomes less significant. Table 6B4 indicates that a value of Z_0 of about $0.5 \mu\text{sec}$ (or around 2 million flops per second) is appropriate. This is low for the published benchmark speed of around 60 million flops per second, and indicates that there may be significant improvements possible with better code or compilers with performance closer to the benchmark speed.

Nevertheless, the estimates of Z_0 are consistent between the different interpolation methods. Even though the Z_0 estimates using linear interpolation are two to four times the Z_0 estimates using Hermite interpolation, it is likely that this is another manifestation of the additional overhead seen in Table 6B1. When using linear interpolation, only 15% of the computational time is consumed in calculating interpolated values. When using Hermite interpolation, in contrast, most of the computational time is consumed in calculating interpolated values in the high-dimension problems. Except for the impact of some unaccounted overhead effort, it seems that equations (6B1) through (6B3) provide good estimates of Z_I and that the interpolation methods are implemented consistently.

Table 6B4. Comparison Between Actual and Hypothetical Growth in Interpolation Effort

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Linear Interpolation</u>							
Eval. interp. (sec.)	.00006	.00006	.00007	.00010	.00020	n.a.	n.a.
Flops per interp.	7	14	27	52	101	198	391
Time/flop (μ sec.)	8.2	4.3	2.6	1.9	2.0	n.a.	n.a.
<u>First-Order Hermite Interpolation</u>							
Eval. interp. (sec.)	.00005	.00014	.00027	.00058	.00155	.00295	.00712
Flops per interp.	26	84	242	680	1862	4948	12770
Time/flop (μ sec.)	1.8	1.7	1.1	.9	.8	.6	.6
<u>Second-Order Hermite Interpolation</u>							
Eval. interp. (sec.)	.00006	.00017	.00043	.00114	.00352	.01084	.03137
Flops per interp.	23	92	362	1352	4662	14932	45026
Time/flop (μ sec.)	2.7	1.8	1.2	.8	.8	.7	.7

3. Growth in Number of Interpolations for Each Discrete State

In each stage, we approximate the cost-to-go function by values at a sufficient number of discrete states. To identify the cost-to-go for each discrete state, we use a search routine that identifies optimal control decisions. Each search for an optimal solution requires a number of cost-function evaluations Z_5 . The total search effort for each discrete state depends on the product of Z_5 and Z_1 to account for overhead effort that includes verification of the solution.

Table 6B5 breaks down the total searches for each interpolation method. By identifying the number of nodes, searches, and interpolations for each of the standardization runs, we can calculate the average number of searches per node and the average number of interpolations per search.

Table 6B5. Breakdown of Effort for Each Discrete State of a Subproblem

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
Discretization, Δ	17	7	5	3	2	2	2
# nodes, Δ^n	17	49	125	81	32	64	128
Linear Interpolation							
# searches	152	428	1128	1128	704	n.a.	n.a.
<i>searches/node</i>	8.9	8.7	9.0	13.9	22.0	n.a.	n.a.
# interpolations	3356	31254	119699	188827	191649	n.a.	n.a.
<i>interp./search</i>	22.1	73.0	106.1	167.4	272.2	n.a.	n.a.
First-Order Hermite Interpolation							
# searches	136	392	1000	648	256	512	1024
<i>searches/node</i>	8.0	8.0	8.0	8.0	8.0	8.0	8.0
# interpolations	522	3107	6985	5062	2095	4690	10179
<i>interp./search</i>	3.8	7.9	7.0	7.8	8.2	9.2	9.9
Second-Order Hermite Interpolation							
# searches	136	392	1000	648	256	512	1024
<i>searches/node</i>	8.0	8.0	8.0	8.0	8.0	8.0	8.0
# interpolations	522	3186	6488	4838	2041	4465	9694
<i>interp./search</i>	3.8	8.1	6.5	7.5	8.0	8.7	9.5

SEARCHES PER NODE FOR EACH INTERPOLATION METHOD

Table 6B5 shows that the minimum number of searches per node is eight. Using limited foresight (Chapter 2), we search for a new set of control decisions for each outcome of the stochastic variables w . Also, the solution for each outcome is verified by at least one restart of the search routine. For the test problems presented in this chapter, we use two discrete values for each stochastic variable, or a total of four discrete outcomes. With at least two calls to a search routine, this requires a minimum of eight searches per node.

The code is written so that each solution is verified by at least one restart of the search routine. The first call is to identify control decisions and optimal cost, and the second is to verify that the solution doesn't change. The restart is especially important for the simplex search routine to verify that the simplex does not degenerate on the first call [Press *et al.*, 1992] and produce an incorrect solution. If the solution is incorrect, a restart should identify a significantly different solution, where significance depends on the tolerance of the simplex method. The restart is also valuable for the quasi-Newton search routine to verify the solution and to ensure good gradient estimates: the first call of the quasi-Newton routine identifies optimal cost and control decisions; the second call verifies the previous solution and evaluates the gradient near the final solution.

Table 6B5 indicates multilinear DDP requires an increasing number of restarts as the dimension of a problem increases. With higher dimension, the simplex appears to have greater opportunity to degenerate, and subsequent restarts are required. The total number of searches increases with dimension at a rate of about $n^{0.5}$. In contrast, the GDP methods require few if any restarts beyond the first, and the number of searches per node does not increase.

INTERPOLATIONS PER SEARCH FOR EACH INTERPOLATION METHOD

Table 6B5 shows that all methods require an increasing number of interpolations per search as the dimension of a problem increases. Multilinear DDP requires 22.1 to 272.2 interpolations for each search using the robust but slow simplex search routine. The number of interpolations increases with dimension at a rate of about $n^{1.5}$. In contrast, the GDP methods require fewer than 10 interpolations for each search, even for the seven-reservoir problem, and the number of interpolations increases with dimension at a rate of about $n^{0.5}$. Differences between the two GDP methods are not great, though it appears second-order method converges slightly more rapidly.

Table 6B5 also shows that the number of interpolations per search for the two-reservoir problem appears break with the general trend, at least when using the Hermite interpolation methods. Following a doubling between the one- and two-reservoir problem, the number drops and increases at a slower rate for the higher-dimension problems.

The computational time can vary considerably with the desired tolerance of the search routine. Because errors of less than about 1% are difficult to achieve with multilinear interpolation (without using finer grids than feasible for the four-reservoir problem), there is little benefit in solving highly accurate function values. As a result, the tolerance of the simplex solver is to a relative error of 10^{-6} . On the other hand, the tolerance of the quasi-Newton solver is to a relative error of 10^{-12} because of the greater accuracy of the Hermite methods and the rapid convergence of the quasi-Newton routine. This additional accuracy also helps to improve gradient estimates. While this extra accuracy entails an increase in computational effort of about 10% for GDP, the increase is about 200% for multilinear DDP.

4. Growth in Total Effort with State Discretization

From the above discussion, we can estimate the total time to evaluate the expected cost-to-go for each node of the state-space grid. The total time per stage is the product of

the time per node and the number of nodes (i.e., discrete states) use to approximate the cost-to-go function.

Each multi-reservoir problem was solved with discretizations up to $\Lambda = 17$ using the different interpolation methods. Not all combinations of dimension, discretization, and interpolation were solved because of the large amounts of time required. Tables 6B6 through 6B8 identify the total standardized time per stage for each of the runs completed. Also included in each table is the average time per node for each run. Unfortunately, it appears that our standardization of computational time does not remove all impacts of different machines and loads. Also, the overhead effort required to execute the code is significant for low dimension problems and coarse discretizations, resulting in higher time for each node.

Table 6B6. Impact of State Discretization on Standard Computational Time of Second-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five
<u>Total Standardized Time for Each Stage (seconds)</u>					
$\Lambda = 2$.31	2.27	5.01	25.51	127
$\Lambda = 3$.44	3.87	19.64	105	802
$\Lambda = 4$.58	6.70	37.81	306	2951
$\Lambda = 5$.79	9.11	72.55	709	9249
$\Lambda = 7$.94	17.59	201	2361	48953
$\Lambda = 9$	1.22	28.31	384	6097	149327
$\Lambda = 13$	1.62	67.19	1083	24547	n.a.
$\Lambda = 17$	2.15	90.07	2474	73949	n.a.
<u>Average Standardized Time for Each Node (seconds)</u>					
$\Lambda = 2$.16	.57	.63	1.59	3.95
$\Lambda = 3$.15	.43	.73	1.29	3.30
$\Lambda = 4$.14	.42	.59	1.19	2.88
$\Lambda = 5$.16	.36	.58	1.13	2.96
$\Lambda = 7$.13	.36	.59	.98	2.91
$\Lambda = 9$.14	.35	.53	.93	2.53
$\Lambda = 13$.12	.40	.49	.86	n.a.
$\Lambda = 17$.13	.31	.50	.89	n.a.

Table 6B7. Impact of State Discretization on Standard Computational Time of First-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Total Standardized Time for Each Stage (seconds)</u>							
$\Lambda = 2$	3.16	2.45	1.21	2.59	7.81	23.04	102
$\Lambda = 3$	2.86	3.33	3.14	10.52	57.66	284	n.a.
$\Lambda = 4$.32	3.58	5.70	33.92	256	1556	n.a.
$\Lambda = 5$.29	4.47	12.95	86.31	737	n.a.	n.a.
$\Lambda = 7$.70	6.89	35.25	332	4593	n.a.	n.a.
$\Lambda = 9$.37	10.52	67.19	923	16090	n.a.	n.a.
$\Lambda = 13$.59	17.34	200	4112	n.a.	n.a.	n.a.
$\Lambda = 17$.66	29.84	453	12248	n.a.	n.a.	n.a.
<u>Average Standardized Time for Each Node (seconds)</u>							
$\Lambda = 2$	1.58	.61	.15	.16	.24	.36	.80
$\Lambda = 3$.95	.37	.12	.13	.24	.39	n.a.
$\Lambda = 4$.08	.22	.09	.13	.25	.38	n.a.
$\Lambda = 5$.06	.18	.10	.14	.24	n.a.	n.a.
$\Lambda = 7$.10	.14	.10	.14	.27	n.a.	n.a.
$\Lambda = 9$.04	.13	.09	.14	.27	n.a.	n.a.
$\Lambda = 13$.05	.10	.09	.14	n.a.	n.a.	n.a.
$\Lambda = 17$.04	.10	.09	.15	n.a.	n.a.	n.a.

Table 6B8. Impact of State Discretization on Standard Computational Time of Second-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Total Standardized Time for Each Stage (seconds)</u>							
$\Lambda = 2$.42	.10	.46	3.06	9.92	87.52	478
$\Lambda = 3$.48	.45	1.62	16.50	82.03	1048	8493
$\Lambda = 4$.53	.94	3.73	29.88	341	5940	64025
$\Lambda = 5$.59	1.30	6.81	126	1647	23100	n.a.
$\Lambda = 7$.68	2.84	19.16	501	8882	n.a.	n.a.
$\Lambda = 9$	1.68	4.66	41.48	1379	31228	n.a.	n.a.
$\Lambda = 13$	1.32	10.61	133	5738	n.a.	n.a.	n.a.
$\Lambda = 17$	1.95	16.69	552	17442	n.a.	n.a.	n.a.
<u>Average Standardized Time for Each Node (seconds)</u>							
$\Lambda = 2$.21	.02	.06	.19	.31	1.37	3.74
$\Lambda = 3$.16	.05	.06	.20	.34	1.44	3.88
$\Lambda = 4$.13	.06	.06	.12	.33	1.45	3.91
$\Lambda = 5$.12	.05	.05	.20	.53	1.48	n.a.
$\Lambda = 7$.10	.06	.06	.21	.53	n.a.	n.a.
$\Lambda = 9$.19	.06	.06	.21	.53	n.a.	n.a.
$\Lambda = 13$.10	.06	.06	.20	n.a.	n.a.	n.a.
$\Lambda = 17$.11	.06	.11	.21	n.a.	n.a.	n.a.

C. ACCURACY OF THE SERIES OF MULTI-RESERVOIR TEST PROBLEMS

As we saw in the last section, finer state discretization increases exponentially the effort to solve each of the seven multi-reservoir test problems. However, in exchange for this additional effort, finer discretization produces more accurate solutions. This section evaluates the accuracy of solutions with the range of discretization presented in the last section.

Accuracy of solutions is evaluated by comparing each approximate cost-to-go function with the “exact” solution. Because we do not know the true exact solution (except when a problem is deterministic or when the condition of certainty equivalence holds), the exact solution is estimated by the most accurate solution available for each reservoir control problem. In all cases, the second-order Hermite solution with the finest discretization is used to estimate the exact solution. For problems with one to four reservoirs, the finest discretization is $\Lambda = 17$; for problems with five to seven reservoirs, the finest discretization is $\Lambda = 9$, $\Lambda = 5$, and $\Lambda = 4$, respectively.

To measure the accuracy of solutions, we evaluate the average absolute relative error between the approximate cost-to-go and the “exact” cost-to-go. At each discrete state identified by the state-space grid of the “exact” solution, the relative error is calculated according to the formula

$$\text{R.E.} = \frac{F(\mathbf{x}^{(i)}) - \text{exact}}{\text{exact}}$$

Where corresponding discrete states do not exist in the approximate solutions, the cost-to-go is interpolated multilinear or Hermite methods as appropriate.

1. Error Reduction with State Discretization

Tables 6C1 through 6C3 display the average absolute relative error (AARE) for the linear and Hermite interpolation methods. Though not shown, the average relative error (ARE) has the same magnitude as the AARE, though ARE is always positive for linear interpolation and is always negative for Hermite interpolation. In other words, linear interpolation overestimates the cost-to-go and Hermite interpolation underestimates the cost-to-go. For linear interpolation, over-estimation is consistent with convexity of the cost-to-go function. For Hermite interpolation, under-estimation is consistent with a cost-to-go function with greater curvature (i.e., larger second derivatives) than used by

the Hermite interpolation. Hermite interpolation uses the lowest curvature consistent with constraints at the boundaries. Indeed, when the true curvature of the cost-to-go function is severe, the Hermite interpolation will oscillate, producing a local minimum that is an artifact of the interpolation.

As expected, error is reduced with finer discretization. Using linear interpolation, AARE is reduced from an average of 300% with $\Lambda = 2$ to about 1% with $\Lambda = 17$. Using Hermite interpolation, AARE is dramatically reduced from an average of 30% with $\Lambda = 2$ to less than 0.002% with $\Lambda = 17$. Accuracy of the second-order Hermite method can be several times greater than accuracy of the first-order method, especially with finer discretization (this is true even if the most-accurate first-order Hermite solutions are used to evaluate the other first-order solutions). Note that first-order and second-order Hermite interpolations produce the same solutions for the one-reservoir problem because the Hessian is a one by one matrix and there are no off-diagonal elements.

With greater accuracy, Hermite interpolation can use coarser state discretizations. If an accurate solution requires an AARE of less than 2%, multilinear DDP requires discretizations finer than $\Lambda = 13$. As a result, it appears impractical to accurately solve problems with four or five state variables using multilinear DDP, except perhaps for problems consisting of only a few stages with no more than four state variables. On the other hand, an AARE of 2% can be achieved by GDP using discretizations as coarse as $\Lambda = 3$. As a result, it appears practical to accurately solve problems with as many as six or seven state variables using GDP.

In Tables 6C1 to 6C3, there is a trend toward somewhat higher error with dimension. For constant values of Λ , the one reservoir problem yields the most accurate solution; though problems with three or more reservoirs have relatively constant errors. Again, solutions for the two-reservoir problem break from the trend. In all cases, the solution error for the two-reservoir problem is significantly greater. This provides clearer evidence of the impact that more complex cost functions have on interpolation accuracy. For such problems, we require finer discretizations; AARE is not reduced below 2% until $\Lambda = 4$ using the second-order Hermite method and until $\Lambda = 5$ using the first-order Hermite method.

Table 6C1. Impact of State Discretization on Accuracy of Linear Interpolation

# Reservoirs, n	One	Two	Three	Four	Five
<u>Average Absolute Relative Error (%)</u>					
$\Lambda = 2$	77	564	216	257	293
$\Lambda = 3$	28	183	72	79	87
$\Lambda = 4$	12	76	34	38	42
$\Lambda = 5$	5.759	40	19	21	23
$\Lambda = 7$	2.778	18	8.028	9.278	10
$\Lambda = 9$	1.518	7.482	4.209	4.913	4.749
$\Lambda = 13$.685	4.760	1.985	2.277	n.a.
$\Lambda = 17$.332	1.522	.908	1.056	n.a.

* second-order Hermite solution with finest discretization used as estimate of exact solution

Table 6C2. Impact of State Discretization on Accuracy of First-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Average Absolute Relative Error (%) (*)</u>							
$\Lambda = 2$	3.0600	61	24	33	54	48	47
$\Lambda = 3$.3829	8.4233	1.4916	1.8646	3.0710	1.5090	n.a.
$\Lambda = 4$.0903	2.6326	.3583	.3955	.6336	.2704	n.a.
$\Lambda = 5$.0276	1.4352	.1408	.1479	.2133	n.a.	n.a.
$\Lambda = 7$.0017	.3178	.0366	.0387	.0485	n.a.	n.a.
$\Lambda = 9$.0012	.1316	.0121	.0141	.0145	n.a.	n.a.
$\Lambda = 13$.0009	.0408	.0048	.0058	n.a.	n.a.	n.a.
$\Lambda = 17$	(*)	.0169	.0017	.0023	n.a.	n.a.	n.a.

* second-order Hermite solution with finest discretization used as estimate of exact solution

Table 6C3. Impact of State Discretization on Accuracy of Second-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Average Absolute Relative Error (%)</u>							
$\Lambda = 2$	3.0600	62	23	31	49	48	46
$\Lambda = 3$.3829	6.1243	1.0534	1.1227	1.4563	1.4962	1.2326
$\Lambda = 4$.0903	1.8740	.2581	.0559	.3196	.2703	(*)
$\Lambda = 5$.0276	1.0048	.0995	.0935	.1134	(*)	n.a.
$\Lambda = 7$.0017	.1943	.0211	.0197	.0209	n.a.	n.a.
$\Lambda = 9$.0012	.0553	.0072	.0071	(*)	n.a.	n.a.
$\Lambda = 13$.0009	.0120	.0012	.0015	n.a.	n.a.	n.a.
$\Lambda = 17$	(*)	(*)	(*)	(*)	n.a.	n.a.	n.a.

* solution with finest discretization used as estimate of exact solution

2. Error Analysis

Kitanidis and Foufoula-Georgiou [1987] compared convergence of Hermite interpolation with that of multilinear interpolation. They demonstrate that with decreases in the discretization interval Δx , the error of the control policy and the cost functions converge as $(\Delta x)^3$ and $(\Delta x)^4$, respectively, using Hermite interpolation versus Δx and $(\Delta x)^2$ using linear interpolation. We can verify that convergence is roughly as expected by considering each pair of solutions using

$$\Lambda = \{(2,3), (3,5), (4,7), (5,9), (7,13), (9,17)\}$$

Since nodes of the state-space grid are evenly spaced, each of these pairs illustrates the improved accuracy obtained by halving the discretization interval. For each pair, we should expect a factor of 4 reduction in error using multilinear DDP and a factor of 16 reduction in error using GDP.

Tables 3C4 through 3C6 display the actual reduction of error. These results are in general agreement with the anticipated reductions in effort. In addition, the reduction in effort for the second-order Hermite method is greater than for the first-order Hermite method. If useful, we could use this result to extrapolate the accuracy of interpolation methods using even finer discretizations.

Table 6C4. Error Reduction Obtained From Halving the Discretization Interval of Linear Interpolation

# Reservoirs, n	One	Two	Three	Four	Five
<u>Ratio of Average Absolute Relative Error</u>					
$\Lambda = 2$ to $\Lambda = 3$	3	3	3	3	3
$\Lambda = 3$ to $\Lambda = 5$	5	5	4	4	4
$\Lambda = 4$ to $\Lambda = 7$	4	4	4	4	4
$\Lambda = 5$ to $\Lambda = 9$	4	5	4	4	5
$\Lambda = 7$ to $\Lambda = 13$	4	4	4	4	n.a.
$\Lambda = 9$ to $\Lambda = 17$	5	5	5	5	n.a.

* second-order Hermite solution with finest discretization used as estimate of exact solution

Table 6C5. Impact of State Discretization on Accuracy of First-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Ratio of Average Absolute Relative Error</u>							
$\Lambda = 2$ to $\Lambda = 3$	8	7	16	18	18	32	n.a.
$\Lambda = 3$ to $\Lambda = 5$	14	6	11	13	14	n.a.	n.a.
$\Lambda = 4$ to $\Lambda = 7$	53	8	10	10	13	n.a.	n.a.
$\Lambda = 5$ to $\Lambda = 9$	23	11	12	10	15	n.a.	n.a.
$\Lambda = 7$ to $\Lambda = 13$	2	8	8	7	n.a.	n.a.	n.a.
$\Lambda = 9$ to $\Lambda = 17$	n.a.	8	7	6	n.a.	n.a.	n.a.

* second-order Hermite solution with finest discretization used as estimate of exact solution

Table 6C6. Impact of State Discretization on Accuracy of Second-Order Hermite Interpolation

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Ratio of Average Absolute Relative Error</u>							
$\Lambda = 2$ to $\Lambda = 3$	8	10	22	27	34	32	37
$\Lambda = 3$ to $\Lambda = 5$	14	6	11	12	13	(*)	(*)
$\Lambda = 4$ to $\Lambda = 7$	53	10	12	13	15	n.a.	n.a.
$\Lambda = 5$ to $\Lambda = 9$	23	18	14	13	(*)	n.a.	n.a.
$\Lambda = 7$ to $\Lambda = 13$	2	16	18	13	n.a.	n.a.	n.a.
$\Lambda = 9$ to $\Lambda = 17$	(*)	(*)	(*)	(*)	n.a.	n.a.	n.a.

* solution with finest discretization used as estimate of exact solution

3. Solutions of the Four-Reservoir Test Problem

To put these measures of error in better perspective, we can view specific solution results. Tables 3C7 and 3C8 display the convergence of the cost-to-go and release decisions for the four-reservoir problem using finer discretization. Because the target releases are below the average inflows, lower initial storage levels ($\mathbf{x}_1 = [1,1,1,1]^T$ versus $\mathbf{x}_1 = [6,6,6,6]^T$) permit lower releases and produce lower expected costs. In both cases, the GDP solutions are accurate with few discrete values (Λ of three or four). Multilinear DDP solutions do not reach a comparable level of accuracy until $\Lambda = 13$ or $\Lambda = 17$. These solutions are in rough agreement with the solutions of *Foufoula Georgiou and Kitanidis* [1988] in spite of a somewhat different model formulation.

Table 6C7. Solution Convergence with Finer Discretization for the Four-Reservoir Problem When $\mathbf{x}_{t_1} = [6,6,6,6]^T$

Λ	Linear					First-Order Hermite				
	F_{t_1}	\mathbf{u}				F_{t_1}	\mathbf{u}			
2	209.22	1.62	2.89	2.28	2.30	42.55	1.63	2.99	2.27	2.69
3	99.29	1.98	2.42	2.42	2.83	66.91	1.45	2.72	1.92	2.47
4	88.00	1.52	2.64	2.10	2.70	68.05	1.49	2.69	2.00	2.51
5	77.44	1.61	2.62	2.13	2.53	68.03	1.47	2.68	1.99	2.53
7	72.48	1.47	2.72	2.00	2.56	68.09	1.48	2.66	2.00	2.52
9	70.63	1.47	2.67	2.08	2.50	68.10	1.49	2.67	2.00	2.52
13	68.91	1.50	2.58	2.01	2.50	68.10	1.49	2.68	2.00	2.52
17	68.62	1.59	2.66	1.98	2.51	68.10	1.49	2.67	2.01	2.52

Table 6C8. Solution Convergence with Finer Discretization for the Four-Reservoir Problem When $\mathbf{x}_{t_1} = [1,1,1,1]^T$

Λ	Multilinear DP					Gradient DP				
	F_{t_1}	\mathbf{u}				F_{t_1}	\mathbf{u}			
2	148.03	1.80	2.12	1.95	.94	-.09	.98	2.50	1.23	.38
3	55.68	1.18	2.16	1.54	.52	9.92	.92	2.38	1.19	.42
4	34.86	.89	2.36	1.20	.50	11.45	.94	2.34	1.19	.42
5	25.04	.82	2.37	1.11	.50	11.67	.94	2.34	1.19	.42
7	17.41	.94	2.40	1.20	.41	11.81	.95	2.33	1.20	.40
9	14.77	.96	2.31	1.14	.42	11.80	.94	2.33	1.19	.40
13	12.87	.96	2.36	1.30	.40	11.83	.94	2.33	1.19	.40
17	12.53	.91	2.36	1.17	.41	11.82	.94	2.33	1.20	.40

D. NET PERFORMANCE OF THE DIFFERENT INTERPOLATION METHODS

We can evaluate the net performance of the different interpolation methods by plotting the accuracy of Tables 6C1-3 against the effort of Tables 6B6-8. The results are presented in Figures 6D1-3. Note that at 10^5 seconds, each stage requires approximately one day of run time. One day per stage is too long for most real-world problems, and the maximum practical time is probably around 10^3 to 10^4 seconds per stage.

As expected, these figures show a trade-off between accuracy and effort. The trade-off using linear interpolation is clearly inferior to the trade-off using either Hermite interpolation method. It is not clear from Figures 6D2 and 6D3 if the first-order Hermite method or the second-order Hermite method is better.

Also, the trade-off between accuracy and effort becomes worse with increasing dimension. In each plot, the trade-off curves shift to the upper right-hand corner with increasing dimension. We again see that the two-reservoir problem breaks from the trend established by the other problems. This is especially true when using first-order Hermite interpolation in Figure 6D2: the trade-off for the two-reservoir problem is worse than for the three-reservoir problem except for the finest discretizations. The break from the trend appears less significant for the second-order method, probably because it includes off-diagonal elements of the Hessian that are significant in the two-reservoir problem.

Figure 6D1. Trade-Off Between Accuracy and Effort per Stage of Linear Interpolation

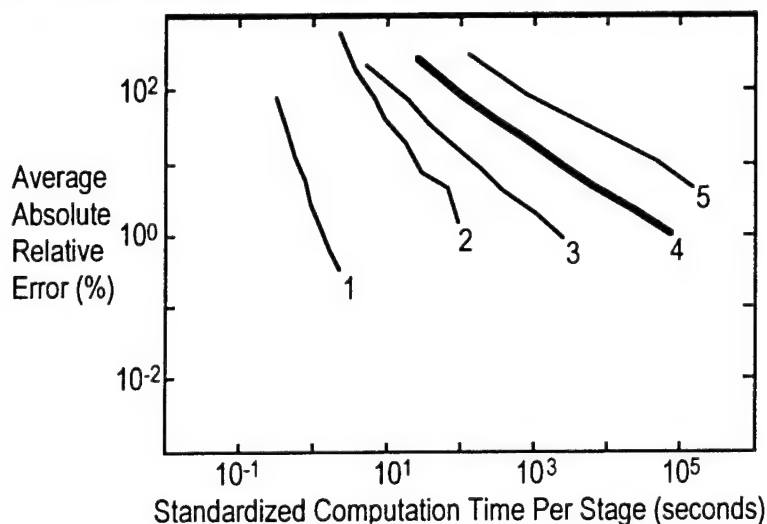


Figure 6D2. Trade-Off Between Accuracy and Effort per Stage of First-Order Hermite Interpolation

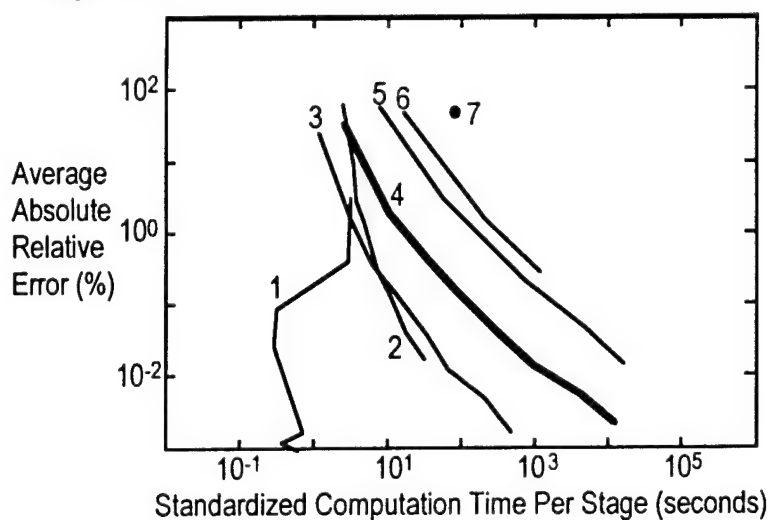
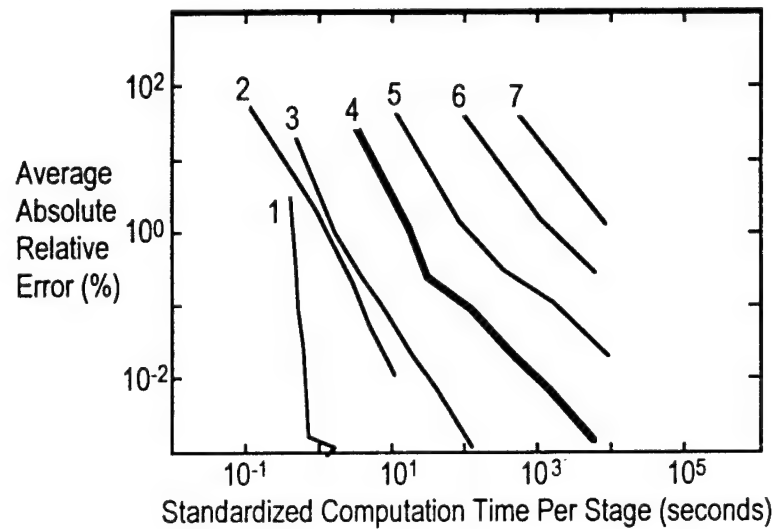


Figure 6D3. Trade-Off Between Accuracy and Effort per Stage of Second-Order Hermite Interpolation



1. A Reformulation of the Results

To better compare the performance of each method, we can identify the amount of time required to achieve a particular level of accuracy. Table 6D1 and Figure 6D4 identify the amount of time required to achieve 1% and 10% AARE using each of the methods. As we can see, using linear interpolation, it can be difficult to solve even the three-reservoir problem when high accuracy and many stages are required. In contrast, using Hermite interpolation, it is relatively easy to solve problems with as many as six state variables. For some problems, it may be possible to include as many as eight state variables.

Figure 6D4 shows that the second-order Hermite method offers the best trade-off between accuracy and effort for low-dimension problems, but the less complicated first-order Hermite method is better for high-order problems. However, as discussed earlier, these test problems are unrealistically simple and off-diagonal elements of the Hessian may be insignificant for all but the two-reservoir problem. For this test problem, the performance of the first-order and second-order Hermite methods are close. In more realistic problems, it seems reasonable to expect that—in spite of the additional effort required—the second-order Hermite method will perform better.

Table 6D1. Standardized Time per Stage to Achieve 10% and 1% Average Absolute Relative Error

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Linear Interpolation Time per Stage (seconds)</u>							
10% AARE	.62	24	170	2200	53000	n.a.	n.a.
1% AARE	1.4	120	2200	80000	n.a.	n.a.	n.a.
<u>First-Order Hermite Interpolation Time per Stage (seconds)</u>							
10% AARE	n.a.	3.0	1.7	4.8	26	56	300
1% AARE	2.8	5.0	3.8	18	180	320	n.a.
<u>Second-Order Hermite Interpolation Time per Stage (seconds)</u>							
10% AARE	n.a.	.33	.65	6.0	27	300	1800
1% AARE	.47	1.3	1.8	18	130	1800	11000

Figure 6D4. Growth in Effort with Number of State Variables to Achieve 10% and 1% Average Absolute Relative Error

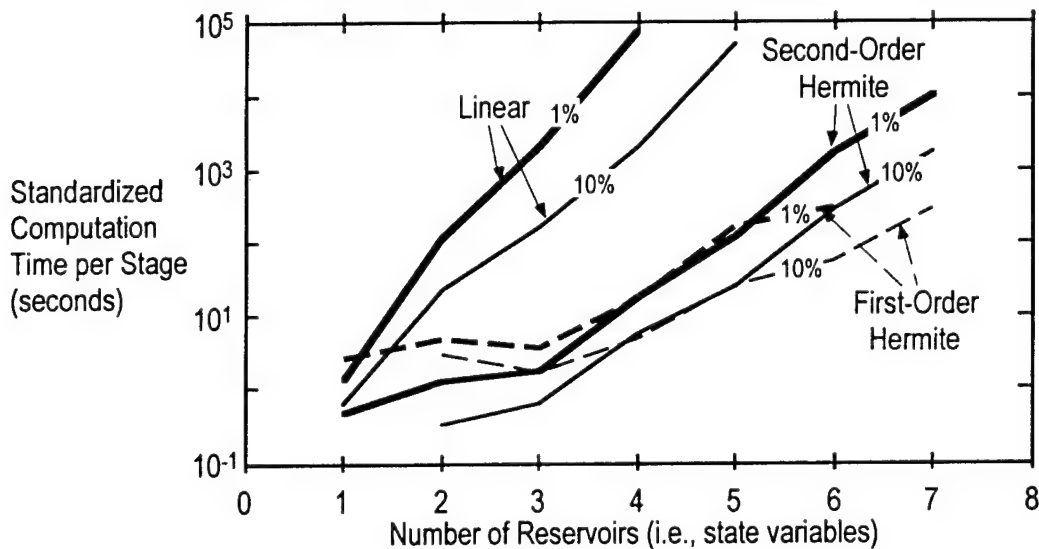


Table 6D2. Ratio of Standardized Time Using Linear and Hermite Interpolation to Achieve 10% and 1% Average Absolute Relative Error

# Reservoirs, n	One	Two	Three	Four	Five	Six	Seven
<u>Linear Versus First-Order Hermite</u>							
10% AARE	n.a.	8.00	100	458	2038	n.a.	n.a.
1% AARE	.50	24.00	579	4444	n.a.	n.a.	n.a.
<u>Linear Versus Second-Order Hermite</u>							
10% AARE	n.a.	72.73	262	367	1963	n.a.	n.a.
1% AARE	2.98	92.31	1222	4444	n.a.	n.a.	n.a.

E. CONCLUDING REMARKS

The results of this Chapter demonstrate the significant computational advantage of GDP methods compared with multilinear DDP. This results from the ability of GDP methods to produce accurate solutions with coarse state discretizations. This also results from the ability to use efficient quasi-Newton search routines that rapidly converge on optimal controls and that provide useful estimates of cost-to-go gradients.

The results are less clear in defining the differences between the two Hermite interpolation methods. The first-order method is better at identifying some solutions and the second-order method is better at identifying other solutions. However, the short time-horizon (i.e., three stages) and the simple cost function (i.e., quadratic and separable, except for the two-reservoir test problem), appears to diminish the value of second derivatives used by the second-order method. In more practical applications, we can expect that the second-order method will perform better. Also, the second-order method is better at preserving convexity of cost-to-go functions. As a result, the second-order method can accurately solve multidimensional problems that the first-order method cannot solve using the same discretization. These expectations are supported by experience solving the conjunctive-use problem of Chapter Eleven.

The results of this Chapter support the application of GDP to reservoir control problems with as many as six to eight state variables. Discretizations as coarse as $\Lambda = 3$ appear sufficient for the test problems presented in this chapter. However, application to practical problems may require finer discretization to achieve the levels of accuracy attained with the test problems. This is especially true when cost-to-go functions have larger changes in curvature that are not easily approximated by third-order polynomials. This is also true when cost off-diagonal elements of the Hessian become more significant, as we saw in the two-reservoir problem. In these cases, it may not be possible to accurately solve problems with more than five or six state variables.

CHAPTER 7.

METHODS OF NUMERICAL INTEGRATION TO EVALUATE EXPECTED VALUES

We have seen that the effort required to solve a DDP problem grows exponentially with the number of state variables. This was explicitly stated in the equation

$$J = Z_I Z_S Z_1 \Lambda^n$$

Total effort per stage grows exponentially with the number of nodes Λ^n and with the effort to find a solution for each node. We have also seen how we reduce the rate of growth by more accurate and efficient interpolation methods.

In the test problems considered in Chapter Six, we avoided the confounding effects of a changing number of stochastic inputs. However, in practice, it is likely that the number of stochastic variables will increase with an increasing number of reservoirs. Larger-scale systems are likely to include numerous inflows that are not perfectly correlated with each other. Also, the number of other stochastic inputs (e.g., water-supply demands, power demands, evaporation and seepage losses, etc.) are likely to increase. In this chapter, we consider the impact that the number of stochastic variables has on computational effort. We also discuss effective methods that can be used to reduce the rate of growth by more accurate and efficient methods of numerical integration.

A. EFFECT OF STOCHASTIC VARIABLE DIMENSION ON COMPUTATION

Up to this point, we have skated around the impact of the dimension of w or s used to model stochastic inputs. However, this is an important component of the "curse of dimensionality" that makes it difficult to solve high dimension stochastic control problems. The effort required to solve stochastic control problems grows exponentially with the number of stochastic variables and entails a trade-off between solution accuracy and computational effort.

1. Effort to Evaluate Expected Values

To evaluate an expected cost-to-go, we apply numerical integration to a number of discrete outcomes of the stochastic variables. Using “limited foresight” of stochastic inputs, we must identify a set of optimal control decisions for each outcome. If each of m stochastic variables has K discrete values, then there are K^m discrete outcomes for each discrete state $\mathbf{x}^{(i)}$.

Note that we cannot avoid this additional effort by identifying “fixed controls” that are feasible for all discrete outcomes (Chapter 2, Section C1). In this case, we replace numerous simple searches with a single difficult search for each discrete state. Fixed controls require K^m evaluations for one iteration of the search while limited foresight requires one evaluation for one iteration. In Chapter Six, we saw that much of the effort of DDP is consumed in evaluating the cost-to-go function. As a result, the total effort is about the same.

At each node $\mathbf{x}^{(i)}$ of the state-space grid we evaluate the expected cost-to-go as

$$F_{t_j}(\mathbf{x}) = \int_{-\infty}^{+\infty} W_{t_j}(\mathbf{w}) [C_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{j+1}}(\mathbf{y})] d\mathbf{w}$$

where $W_{t_j}(\mathbf{w})$ is the probability density function for the stochastic input \mathbf{w} . State \mathbf{y} at the end of a stage is a result of \mathbf{w} , control decisions \mathbf{u} , and initial state \mathbf{x} . In general, we cannot analytically evaluate this integrand. Instead, we approximate the expected cost as a probability weighted sum

$$\int_{-\infty}^{+\infty} W_{t_j}(\mathbf{w}) [C_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{j+1}}(\mathbf{y})] d\mathbf{w} \approx \sum_{k=1}^{K^m} \{ v_k [C_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}^{(k)}) + F_{t_{j+1}}(\mathbf{y}^{(k)})] \}$$

where m is the number of stochastic variables and K is some average number of discrete values used to span each stochastic variable.

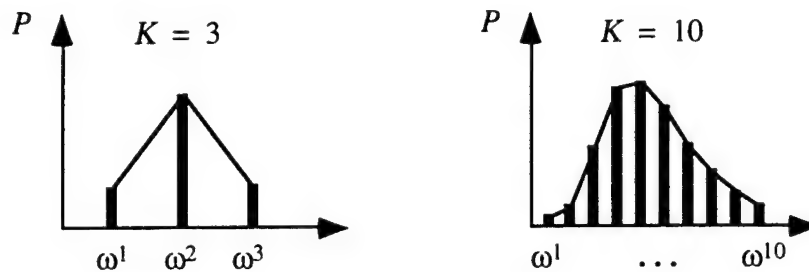
The effort required to calculate the expected future cost is proportional to the factor K^m . Thus, the effort required to solve a DDP problem can be represented by the equation

$$J = Z_I Z_S Z_2 K^m \Lambda^n \quad (7A1)$$

where the term K^m explicitly breaks out the number of searches required to calculate the expected value as a probability weighted summation. Z_2 is the remaining overhead effort.

Just as it is desirable to mitigate the exponential growth in effort with number of state variables, it is also desirable to mitigate the exponential growth in effort with number of stochastic variables. By using efficient numerical integration techniques, we can greatly reduce the number of stochastic nodes K^m required to accurately solve a Stochastic DP problem. For example, Figure 7A1 illustrates that a crude method of numerical integration may result in significant error. When used to evaluate the area under a curve, we may need a fine discretization. This error can be especially large when applied in a DDP algorithm because the highest costs are often associated with the tails of a distribution, where a crude method of numerical integration does poorly.

Figure 7A1. Probability Distributions for Various Discretizations of a Stochastic Variable



2. Evaluation of Expected Values by Numerical Integration

Numerical integration, also called quadrature, has long provided methods for the integration of functions that could not easily be computed analytically [Press *et al.*, 1992, Chapter 4]. Numerical integration is the approximation of an integral by a weighted summation

$$\int_a^b g(z) dz \approx \sum_{k=1}^M v_k g(z^{(k)}) \quad (7A2)$$

where the goal is to accurately obtain an accurate integral value with as few function evaluations $g(z^{(k)})$ as possible. This is accomplished by use of a sufficiently fine discretization interval between abscissas $z^{(k)}$ and by selection of an appropriate method to assign weights v_k .

In a DDP problem with one stochastic variable, the abscissas are the discrete outcomes $z^{(k)}$ of a stochastic variable. In DDP problems with multiple stochastic variables, the abscissas are nodes of a “stochastic space” grid defined by the discrete values of each stochastic variable. These nodes identify all possible outcomes $\mathbf{z}^{(k)}$ for combinations of the discrete stochastic values, just as the nodes of the state-space grid

identify all possible outcomes $\mathbf{x}^{(i)}$ for combinations of the discrete state values. The weight applied to each outcome, or node, is the product of the weights for the discrete values,

$$v_k = \prod_{j=1}^m v_{k,j}$$

assuming the stochastic variables are independent.

3. Classical Numerical Integration: The Trapezoidal Rule

The most common and straightforward numerical integration method is the trapezoidal rule. In essence, the trapezoidal rule uses linear interpolation between discrete outcomes to evaluate the area under a curve (Figure 7A1). In other words, numerical integration over each interval is estimated by equation (7A2) with weights

$$v_k = h/2, \quad z^{(k)} \in \{a, b\}$$

$$v_k = h, \quad \text{otherwise}$$

where $h = (K-1) / (b-a)$ using K equally spaced abscissas. Using unequally spaced abscissas, the weights are

$$v_k = \frac{1}{2(z^{(k+1)} - z^{(k)})}, \quad z^{(k)} = a$$

$$v_k = \frac{z^{(k+1)} - z^{(k-1)}}{2(z^{(k+1)} - z^{(k)})(z^{(k)} - z^{(k-1)})}, \quad a < z^{(k)} < b$$

$$v_k = \frac{1}{2(z^{(k)} - z^{(k-1)})}, \quad z^{(k)} = b$$

The error is $\sim O((b-a)^3 f'')$ where f'' is the second derivative somewhere in each interval of integration [Press *et al.*, 1992, pp. 125-6]. The trapezoidal rule is the starting point in the development of many other quadrature methods.

4. Examples of Past Efforts to Evaluate Expected Values

To evaluate expected values, past stochastic dynamic programming efforts have often used a form of the trapezoidal rule called the extended midpoint rule. This rule locates the abscissa halfway between the endpoints of each interval [Press *et al.*, 1992, p. 129] and evaluates the expected value as the probability weighted average cost of all abscissas:

$$\int_{-\infty}^{+\infty} \{ W(s) V(s) \} ds \approx \sum_{k=1}^M \{ W(s^{(k)}) \Delta s V(s^{(k)}) \}$$

where $v_k = W(s^{(k)}) \Delta s$ is the probability weight assigned to abscissa s_k . The error of these various approaches is $\sim O((\Delta s)^3 (WV)'')$ where $(WV)''$ is the second derivative of the product of $W(s)$ and $V(s)$ somewhere in the interval containing $s^{(k)}$.

Most commonly, equal intervals have been used to discretize the stochastic variable. For example, *Turgeon* [1981] uses ten equally spaced inflows in an aggregate reservoir. *Weiner and Ben-Zvi* [1982] use nine discrete inflows in a "one-reservoir" hydropower model using hypothetical flows from the Mediterranean to the Dead Sea. *Karamouz and Houck* [1987] use from two to nine intervals to evaluate the impact on solution accuracy. *Valdes et al.* [1992] discretize stochastic variables using a method by *Bras et al.* [1983] with conditional probabilities that depend on prior inflow. They apply the method to a one-reservoir hydropower model using five discrete inflows. *Raman and Chandramouli* [1996] use seven to thirty-five discrete inflows in DDP to contrast the performance of a neural-network model. *Esmail Beik and Yu* [1984] represent a distribution by 8 to 28 net inflows specified in advance. These inflows are selected "to change storage by exactly a multiple of the [storage] interval with a one-to-one correspondence between the resulting storage and the prescribed states." This avoids the need for interpolation between nodes of the state-space grid. Weights on each inflow are conditioned on release decisions and prior inflows.

Equiprobable intervals have also been used to discretize stochastic variables. For example, *Kelman et al.* [1990] use a large, unspecified number of equally weighted scenarios. *Karamouz and Vasiliadis* [1992] develop a method that divides the distribution of a stochastic variable into a number of equally probable values. They apply the method to a one-reservoir water supply model using eight flow values. When using equiprobable intervals, abscissas may be placed at locations other than the midpoints.

In some cases, uneven intervals are selected as somehow representative of the problem at hand. *Stedinger et al.* [1984] use abscissas located at the 97.5, 83.125, 59.375, 35.625, and 11.875 percentiles of the unconditional probability distributions, and updated the weights using prior streamflows. *Tejada-Guibert et al.* [1993] apply a method by *Max* [1960] to identify five discrete inflows that are partitioned into flows for two hydropower reservoirs.

When analyzing large-scale systems, quadrature accuracy is sacrificed to allow detailed system models. Often, two discrete values are used to represent a stochastic

variable [Gorenstin *et al.*, 1992; Pereira and Pinto, 1985; Pereira and Pinto, 1991; Rotting and Gjelsvik, 1992]. Johnson *et al.* [1993] approximate lognormal distributions using abscissas at the 5, 50, and 95 percentiles and weights of 1/6, 2/3, and 1/6. Jacobs *et al.* [1995] use sorting of historical flows to define three discrete values that represent high, median, and low flow scenarios. In these examples, the selection of abscissas and weights was heuristic and it appears that quadrature accuracy was not considered.

One exception is the application of quadrature by Fouloula Georgiou and Kitaniadis [1988]. They apply DDP to a problem with two stochastic variables and only two discrete values for each. Using Gauss-Hermite quadrature, they achieve a high degree of accuracy.

5. Discretization of Stochastic Variables in Past Efforts

In all examples above, either fine discretization is used to model the distribution of stochastic variables, or the impact of discretization on solution accuracy is ignored. Tejada-Guibert *et al.* [1995] evaluate the benefit of hydropower generation, noting that “the discrete inflow approximation typically employed in SDP models often does not provide good resolution of hydrologic extremes. If the objective function employed is sensitive to extremes, as it is when large penalties are placed on shortages or damages due to flooding are considered, the deviations of the SDP-generated gains from the simulated gains are likely to be large and thus not be a good guide for reservoir release decisions and other water and energy related marketing decisions.” While this observation identifies a common problem with many past efforts, the problem is not inherent to stochastic dynamic programming but to the quadrature methods employed.

When the objective function is “sensitive to extremes” (i.e., when the objective function is not quadratic), numerous discrete values may be required to “provide good resolution of hydrologic extremes.” Fine discretization of stochastic variables is required to evaluate expected values with accuracy just as fine discretization of state variables is used to approximate cost-to-go functions with accuracy. However, this is possible only with one or two stochastic variables. When multiple stochastic variables are needed to model a system, the solution effort grows almost as fast as the solution effort for multiple state variables. The solution effort is somewhat less only because the number of stochastic variables does not increase interpolation and solver effort (Z_I and Z_S in Chapters Five and Six).

The effort required to calculate expected values, especially when using multiple stochastic variables, does not appear to have been a significant consideration in past efforts. In the examples cited, the efficiency of quadrature has not been a focus of effort.

Esmail Beik and Yu [1984] present one of the few examples of growing computational effort with increasing discretization, but they use only a single stochastic variable. In large part, past efforts may have been so limited in their consideration of problems with multiple state variables that concern for the impact of multiple stochastic variables was limited.

In problems with one stochastic variable [*Esmail Beik and Yu*, 1984; *Karamouz and Houck*, 1987; *Karamouz and Vasiliadis*, 1992; *Kelman et al.*, 1990; *Raman and Chandramouli*, 1996; *Saad et al.*, 1996; *Stedinger et al.*, 1984; *Tejada-Guibert et al.*, 1993; *Valdes et al.*, 1992; *Weiner and Ben Zvi*, 1982] fine discretizations have been used. In these cases, no fewer than five discrete values have been used to accurately estimate expected values.

In problems with multiple stochastic variables [*Gorenstin et al.*, 1992; *Jacobs et al.*, 1995; *Johnson et al.*, 1993; *Pereira and Pinto*, 1985; *Pereira and Pinto*, 1991; *Rotting and Gjelsvik*, 1992; *Turgeon*, 1981], only two or three discrete values are identified for each stochastic variable. In these cases, the quadrature accuracy is sacrificed to allow a detailed system model. As mentioned, the one exception to this is the application Gauss-Hermite Quadrature by *Foufoula Georgiou and Kitanidis* [1988]. The next section discusses the advantage of Gaussian quadrature and specific methods that are appropriate for reservoir management problems.

B. THE APPLICATION OF GAUSSIAN QUADRATURE TO DISCRETE DYNAMIC PROGRAMMING

In "classical" methods of quadrature, numerical integration is applied first by identifying abscissas using some heuristic method (usually evenly spaced) and second by identifying appropriate weights. The idea of Gaussian quadrature is evaluate abscissas and weights together. In effect, this doubles the degrees of freedom that we have to estimate an integrand accurately [*Press et al.*, 1992, p 140]. When an integrand is very smooth (i.e., well approximated by a polynomial), Gaussian quadrature allows us to achieve levels of accuracy using only half the number of abscissas of classical high-order methods. Moreover, when an integrand fits certain useful forms, Gaussian quadrature is exact.

Bellman and Dreyfus [1962, pp. 324] first proposed the use of quadrature as a method to circumvent the difficulty in evaluating integrals in dynamic programming. They propose Gaussian quadrature to calculate the coefficients of polynomials used to approximate the cost-to-go function. They used a parametric dynamic programming

method to reduce the exponential growth in storage with higher state dimension, and the expected value function fit reasonably well the form used by Gaussian quadrature. To apply Gaussian quadrature in DDP has required cost-to-go approximations that provide similar smoothness. Recent developments of high-order Hermite and spline interpolation methods now offer this degree of smoothness and allow application of Gaussian-quadrature methods.

1. Mathematical Form of Gaussian Quadrature

In equation (7A2), Gaussian quadrature assigns not only the weights v_k to be applied, but also assigns the locations z_k of the abscissas. By carefully choosing the location of abscissas and weights, we can obtain higher-order accuracy than possible by quadrature methods that rely on either equally-spaced or arbitrarily-spaced abscissas.

Of additional importance, for a certain class of integrands, "polynomials times some known weighting function," numerical integration can be exact. This class of integrands can be described by the equation

$$\int_a^b W(z) g(z) dz \approx \sum_{k=1}^K v_k g(z^{(k)}) \quad (7B1)$$

where $g(z)$ is a polynomial and $W(z)$ is some known weighting function. The restrictions are that (1) $W(z)$ be sufficiently smooth or from the known set of weighing functions for which Gaussian quadrature is exact, and (2) we use a sufficient number of abscissas. The number of abscissa is sufficient if it provides degrees of freedom greater than or equal to the number of unknowns. In other words, if we use K abscissas, quadrature is exact for integrands with arbitrary polynomials of degree $(2K - 1)$ or less.

2. Compatibility of Gaussian Quadrature and Hermite Interpolation

Using the Hermite interpolation of equations (5C1) or (5H1), the cost-to-go $F_{t_{j+1}}$ is a piecewise third-order polynomial function in each dimension of y . If the cost-to-go function is discretized with a sufficiently coarse grid, the apparent cost-to-go for various outcomes of y is adequately described by a third-order polynomial, and the piecewise nature of $F_{t_{j+1}}$ will not have a significant impact.

If we consider a system with linear dynamics, no constraints, and a current cost function C_{t_j} represented by a third-order polynomial, then the total cost function given by equation (4A5)

$$V_{t_j} = C_{t_j}(x, u, w) + F_{t_{j+1}}(y)$$

is adequately described as third-order polynomial function of each random normal variable w_k . For initial state \mathbf{x}^i and fixed decisions \mathbf{u} , we can identify the expected cost by

$$F_{t_j}(\mathbf{x})|\mathbf{u} = E_{\mathbf{w}}\{ V_{t_j} \} = \int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} \{ W(\mathbf{w}) V_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}) \} d\mathbf{w}_1 \dots d\mathbf{w}_m \quad (7B2)$$

This equation fits the form of equation (7B1) for each variable of \mathbf{w} evaluated independently. $W(\mathbf{w})$ is the probability density function for a normal distribution. As we will see, this is in the known set of weighting functions for Gaussian quadrature, and Gaussian quadrature is exact with only two discrete values for each random variable (i.e., $K = 2$).

In modeling a system, we may transform a stochastic input \mathbf{s} to obtain independent random normal variables. In such cases, the transition function probably is not a linear function of \mathbf{w} and V_{t_j} is not a third-order polynomial function of \mathbf{w} . Instead, the expected cost can be expressed as a function of \mathbf{s} to preserve the form of equation (7B1):

$$F_{t_j}(\mathbf{x})|\mathbf{u} = E_{\mathbf{s}}\{ V_{t_j} \} = \int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} \{ W(\mathbf{s}) V_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{s}) \} d\mathbf{s}_1 \dots d\mathbf{s}_m \quad (7B3)$$

If the distribution $W(\mathbf{s})$ is sufficiently smooth or comes from the known set of weighting functions, Gaussian quadrature applied to equation (7B3) is exact.

It is useful to note that we may still apply Gaussian quadrature (or, more specifically, Gauss-Hermite quadrature) using equation (7B2) even when if a transformation of \mathbf{s} to \mathbf{w} results in a non-linear transition function and less smooth total-cost function. Most transformations of \mathbf{w} do not make V_{t_j} significantly less smooth; and V_{t_j} may still be appropriately described as a third-order polynomial function of \mathbf{w} . As a result, Gaussian quadrature applied to equation (7B2) will be accurate, or may be made sufficiently accurate with the addition of one or more abscissas.

3. Sources of Error in Applying Gaussian Quadrature to Discrete Dynamic Programming

The integrand used to evaluate the expected cost-to-go has a reasonable fit to the form required by Gaussian quadrature, but the fit is not exact. As a result, there are various sources of error that may result in poor quadrature. While none of these invalidate the use of Gaussian quadrature, the accuracy of solutions should be verified.

We will assess the impact of these sources of error in Chapter Eight. The following summarizes sources of error and conditions under which errors may be significant.

PIECEWISE NATURE OF COST-TO-GO APPROXIMATIONS

Gaussian quadrature is exact if applied to integration of a polynomial times an appropriate weighting function. In application to DDP, Gaussian quadrature is approximate because the interpolated cost-to-go function is divided into a number of subdomains. Even though each subdomain is described by an n -fold third-order polynomial function, the entire domain is described by a piecewise fitting of these functions that, in its entirety, may not be an n -fold third-order polynomial function. However, when evaluating an expected value using quadrature, the outcomes may sample a portion of the domain that can be approximated by some n -fold third-order polynomial function (or other n -fold low-order polynomial).

The piecewise nature of cost-to-go approximations can have a variable impact on quadrature accuracy. On one hand, finer subdivision of the state space can degrade quadrature accuracy as outcomes sample more subdomains in a portion of the domain. The cost-to-go function may appear more complex and accurate quadrature may require finer stochastic discretization to evaluate a higher-order polynomial form. On the other hand, finer subdivision of the state space can improve quadrature accuracy as adjacent subdomains use interpolating functions that are more consistent. Also, finer subdivisions reduce the likelihood that interpolation error will destroy the higher-order accuracy of Gaussian quadrature. This can be a large problem when coarse state discretization permits oscillation in the interpolating functions.

CHANGING CONTROL DECISIONS

We solve control decisions for each outcome of \mathbf{w} using limited foresight (equation (4A6)):

$$F_{t_i}(\mathbf{x}) = E_{\mathbf{w}}\{ \min_{\mathbf{u}}\{ V_{t_i} \} \}$$

As a result, control decisions are not fixed, and the expected total-cost function may not fit the form of equation (7B1). Fortunately, however, control decisions usually vary smoothly when not at a bound, and the resulting cost as a function of \mathbf{w} is almost as smooth. Thus, Gaussian quadrature will often be sufficiently accurate without an increase in the number of abscissas.

NON-POLYNOMIAL COST AND TRANSITION FUNCTIONS

Gaussian quadrature is exact if integration is applied to equations that fit the form of equation (7B1). As discussed above, interpolation of the cost-to-go function results in deviations from this form. However, non-polynomial current-cost and transition functions also result in deviations from this form, and may require additional abscissas to achieve sufficient quadrature accuracy.

Fortunately, transition functions for many reservoir management problems are linear. Also, the summation of the current-cost and the cost-to-go will increase smoothness of the total cost, especially near optimal solutions. In finding the optimal trade-off between current and future costs, control decisions will generally avoid the most-curved, high-cost regions of a non-polynomial cost function. For example, in Chapter Ten, we will develop a cost function that becomes infinite as water supplies approach zero. Though this creates significant problems for quadrature when applied at zero supply, optimal control decisions are cautious and avoid this condition.

CONSTRAINTS

Gaussian quadrature assumes that the cost function is an unbounded polynomial function of the stochastic variables s . In application to reservoir problems, various constraints on system operation can produce costs that are a more complex function of s . For example, high inflows may fill a reservoir, requiring large downstream releases producing extreme costs. In this case, a constraint on reservoir level is binding, and the form of the cost function (as a function of inflows) may change significantly.

When stochastic inputs are described by continuous distributions (as in the case of reservoir inflows), costs change smoothly even when constraints becoming binding. When using quadrature, changes may not be smooth, but may kink as each outcome shifts from non binding to binding (e.g., as we consider different initial states or as a solver iteratively changes control decisions). This may increase quadrature error and the number of iterations required by a solver for convergence. These potential problems can be addressed by using finer state discretization and finer stochastic discretization.

Finer state discretization improves interpolation accuracy, especially near boundaries of the state-space. This can significantly improve the smooth transition of costs as constraints change from non-binding to binding at the boundary. Specifically, accurate interpolation of gradients at a boundary prevents state variables from becoming binding until marginal costs (with respect to control decisions) near a boundary match marginal costs at that boundary.

Finer stochastic discretization improves quadrature accuracy and reduces the sharpness of kinks as each outcome shifts from non binding to binding. With finer discretization, the weight applied to each outcome is less. Note, however, that the structure of DDP works in our favor to reduce the impact of kinks in a cost function. The first is that summation of current and future costs smooth out an estimate of the total-cost function, as discussed earlier. The second is that interpolation of a cost-to-go function smoothes out any kinks that may exist from prior subproblems.

C. IDENTIFICATION OF GAUSSIAN QUADRATURE WEIGHTS AND ABSCISSAS

Gaussian quadrature is exact when applied to integration of polynomials times some known weighting function. The previous section considered how well an interpolated cost function fits a polynomial form. This has significance for the appropriate application of Gaussian quadrature to DDP. This section considers how well stochastic variables such as inflow fit appropriate weighting functions. This has significance for the appropriate application of Gaussian quadrature to reservoir management problems.

Stochastic inputs in reservoir management problems are usually described by continuous and smooth probability distributions that are well suited for Gaussian quadrature. For example, streamflow, demand, loss from evaporation or seepage, and other stochastic "inputs" are often described by normal, lognormal, or related distributions. In this section, we will consider "known weighting functions" that are appropriate for reservoir management problems.

1. Gaussian Quadrature with Normal Distributions

Normal distributions are often used to model stochastic inputs because they produce models with convenient mathematical properties. These properties may allow simpler solution of systems analysis problems, such as when the condition of certainty equivalence holds or when we use first-order analysis (Chapter Three). Normal distributions also are simple and their use is parsimonious. In other words, a simple model is more appropriate when data are insufficient to justify a more complex model.

Weighting functions based on normal distributions fit the quadrature form known as Gauss-Hermite quadrature [Press *et al.*, 1992, p. 144]. Gauss-Hermite quadrature uses the weighting function

$$W(z) = e^{-z^2}$$

Numerical integration of the form

$$\int_{-\infty}^{+\infty} e^{-z^2} g(z) dz = \sum_{k=1}^K v_k g(z^{(k)}) \quad (7C1)$$

can be solved exactly for any polynomial function $g(z)$.

In application to normally distributed stochastic input s described by mean μ and variance σ^2 , we scale abscissas and weights of Gauss-Hermite quadrature. The one-dimensional form of equation (7B3) can be expressed using normally distributed inputs as

$$E_S\{ V_{t_j} \} = \int_{-\infty}^{+\infty} (\sigma\sqrt{2\pi})^{-1} e^{-(s-\mu)^2/2\sigma^2} V_{t_j}(s) ds \quad (7C2)$$

If we substitute $s = z\sigma\sqrt{2} + \mu$ into equation (7C2), we get

$$E_S\{ V_{t_j} \} = \int_{-\infty}^{+\infty} \pi^{-1/2} e^{-z^2} V_{t_j}(z\sigma\sqrt{2} + \mu) dz \approx \pi^{-1/2} \sum_{k=1}^K v_k V_{t_j}(z^{(k)}\sigma\sqrt{2} + \mu)$$

where $w = z\sqrt{2}$ is a random variable with standard-normal distribution. The abscissas and weights for s are the scaled values $z^{(k)}\sigma\sqrt{2} + \mu$ and $v_k/\sqrt{\pi}$. Table 7C1 identifies the scaled abscissas and weights for a $\sim N(0,1)$ distribution (i.e., standard-normal) with up to $K = 9$.

Table 7C1. Gaussian Quadrature Abscissa Locations and Weights for Standard Normal Distribution

	Abscissa and (Weight)									
$K = 1$	0 (1.0)									
$K = 2$	-1.00 +1.00 (.5) (.5)									
$K = 3$	-1.73 0 +1.73 (.16667) (.66667) (.16667)									
$K = 4$	-2.33 -0.74 +0.74 +2.33 (.04588) (.45412) (.45412) (.04588)									
$K = 5$	-2.86 -1.36 0 +1.36 +2.86 (.01126) (.22208) (.53333) (.22208) (.01126)									
$K = 6$	-3.32 -1.89 -0.62 +0.62 +1.89 +3.32 (.00256) (.08862) (.40883) (.40883) (.08862) (.00256)									
$K = 7$	-3.75 -2.37 -1.15 0 +1.15 +2.37 +3.75 (.00055) (.03076) (.24012) (.45714) (.24012) (.03076) (.00055)									
$K = 8$	-4.14 -2.80 -1.64 -0.54 +0.54 +1.64 +2.80 +4.14 (.00011) (.00964) (.11724) (.37301) (.37301) (.11724) (.00964) (.00011)									
$K = 9$	-4.51 -3.21 -2.08 -1.02 0 +1.02 +2.08 +3.21 +4.51 (.00002) (.00279) (.04992) (.24410) (.40635) (.24410) (.04992) (.00279) (2.2E-5)									

2. Gaussian Quadrature with Three-Parameter Gamma Distributions

When sufficient data is available to use more complex models, it may be useful to fit the quadrature form known as Gauss-Laguerre [Press *et al.*, 1992, p. 144]. Gauss-Laguerre quadrature uses the weighting function

$$W(z) = z^a e^{-z}$$

where parameter a is a third parameter that may be used to fit a distribution. Numerical integration of the form

$$\int_0^{+\infty} z^a e^{-z} g(z) dz \approx \sum_{k=1}^K v_k g(z^{(k)}) \quad (7C3)$$

can be solved exactly for any polynomial function $g(z)$.

Gamma distributions are appropriate for modeling a variety of hydrologic processes. For example, the Pearson Type III distribution is a gamma distribution [Bras, 1990; Wallis *et al.*, 1974]. Gauss-Laguerre quadrature allows modeling of stochastic variables that are strictly positive and have significantly skewed distributions. This is

especially appropriate in application to hydrologic models that often incorporate non-negative, skewed stochastic variables for streamflow and demand.

3. Gaussian Quadrature with Lognormal Distributions

Hydrologic variables are frequently modeled using lognormal distributions. However, lognormal distributions do not fit the form of known weighting functions for which abscissas and weights have been solved. As suggested earlier, one approach to applying Gaussian quadrature with lognormal distribution is to apply quadrature to the transformed variables w , and add sufficient abscissas to achieve the desired accuracy. This approach still should produce higher-order accuracy than obtained using the trapezoidal rule or using a heuristic method.

When the logarithm of a lognormal stochastic input s is described by mean μ and variance σ^2 , the one dimensional form of equation (7B3) can be expressed as

$$E_s\{ V_{ij} \} = \int_{-\infty}^{+\infty} (s\sigma\sqrt{2\pi})^{-1} e^{-(\ln(s)-\mu)^2/2\sigma^2} V_{ij}(s) ds \quad (7C4)$$

If we substitute $s = \text{Exp}[z\sigma\sqrt{2} + \mu]$ into equation (7C3), we get

$$E_s\{ V_{ij} \} = \int_{-\infty}^{+\infty} \pi^{-1/2} e^{-z^2} V_{ij}(e^{z^{(k)}\sigma\sqrt{2}+\mu}) dz \approx \pi^{-1/2} \sum_{k=1}^K v_k V_{ij}(e^{z^{(k)}\sigma\sqrt{2}+\mu})$$

Thus, the abscissas and weights are the scaled values $s^{(k)} = \text{Exp}[z^{(k)}\sigma\sqrt{2} + \mu]$ and $v_k / \sqrt{\pi}$. Notice that the cost function is a complex non-polynomial function, and that Gaussian quadrature may require additional abscissas to accurately identify the expected cost.

4. Gaussian Quadrature with Arbitrary Distributions

Instead of applying Gauss-Hermite quadrature to transformed variables w , we can use a general approach to identify abscissas and weights for any distribution that is sufficiently smooth. As a result, we can apply Gaussian quadrature without degrading the accuracy by a non-linear transformation. The following presents a practical method for identifying abscissas and weights for Gaussian quadrature when using an arbitrary weighting function $W(z)$. The method and equations are taken from *Press et al.* [1992, pp. 142-4], with some adaptation of notation to that used in this thesis.

Define the scalar product

$$\langle f | g \rangle \equiv \int_a^b W(z) f(z) g(z) dz \quad (7C5)$$

and the series of polynomial functions

$$p_{-1}(z) \equiv 0$$

$$p_0(z) \equiv 1$$

$$p_{j+1}(z) = (z - a_j) p_j(z) - b_j p_{j-1}(z), \quad j = 0, 1, 2, \dots$$

where

$$a_j = \frac{\langle z p_j | p_j \rangle}{\langle p_j | p_j \rangle}, \quad j = 0, 1, 2, \dots$$

$$b_j = \frac{\langle p_j | p_j \rangle}{\langle p_{j-1} | p_{j-1} \rangle}, \quad j = 1, 2, 3, \dots$$

For an arbitrary weighting function $W(z)$ applied over an interval $[a, b]$, the K abscissas for Gaussian quadrature are the roots of the $p_K(z)$ polynomial. To identify roots of the $p_K(z)$ polynomial, it is useful to progressively identify the roots of $p_1(z)$, $p_2(z)$, ..., $p_K(z)$, since the roots of each polynomial $p_j(z)$ interleave the roots of each polynomial $p_{j-1}(z)$. The weights for each abscissa z_j can be identified by the formula

$$v_j = \frac{\langle p_{K-1} | p_{K-1} \rangle}{p_{K-1}(z_j) q_K(z_j)}$$

where $q_K(z_j)$ is the derivative of $p_K(z)$ at z_j . Tables 7C2 through 7C5 identify the abscissas and weights for a variety of lognormal distributions. We will apply these abscissas and weights to problems in Chapter Eight to evaluate the performance of Gaussian quadrature.

Practical techniques to evaluate these polynomials and their roots may require some trial and error. In this thesis, the roots were identified using the mathematical solver, *Mathematica* [Wolfram, 1991]. It was observed that convergence of numerical integration applied to equation (7C5) was poor when using unbounded distributions or distributions for which the lower bound is zero (e.g., lognormal distributions). In these cases, it was useful to use bounds that were slightly greater than zero and sufficiently large but not infinite. Convergence was found to be especially slow for high-order polynomials (e.g., ninth-order or larger), and numerical error also may be significant. However, practical application should generally require evaluation of only the second and third-order polynomials. Somewhat higher-order polynomials may also be desired to verify the accuracy using these coarse stochastic discretizations.

Unbounded highly skewed distributions can produce abscissas that are too large for a model. This is especially true when models include prior values of stochastic variables to incorporate autocorrelation. Under these conditions, it is practical to use more restrictive bounds $[a,b]$ since roots of the polynomials all lie within this interval.

Table 7C2. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(2.0, 0.5)$

Abscissa and (Weight)	
$K = 1$	2 (1.0)
$K = 2$	1.66 2.73 (.67876)(.32124)
$K = 3$	1.46 2.26 3.49 (.39360)(.56148)(.04492)
$K = 4$	1.33 1.99 2.89 4.32 (.22328)(.60860)(.16449)(.00364)
$K = 5$	1.24 1.81 2.55 3.59 5.25 (.12853)(.54940)(.29685)(.02502)(.00019)
$K = 6$	1.16 1.67 2.31 3.17 4.38 6.31 (.07601)(.45609)(.39526)(.07033)(.00230)(6.9E-6)
$K = 7$	1.10 1.57 2.14 2.88 3.87 5.27 7.51 (.04636)(.36379)(.44812)(.13195)(.00964)(.00014)(1.8E-7)
$K = 8$	1.05 1.49 2.01 2.66 3.51 4.66 6.28 8.87 (.02917)(.28509)(.46261)(.19758)(.02471)(.00084)(5.8E-6)(3.4E-9)

Table 7C3. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(4.0, .75)$

Abscissa and (Weight)	
$K = 1$	4 (1.0)
$K = 2$	3.43 4.99 (.63684)(.36316)
$K = 3$	3.10 4.29 5.93 (.33049)(.60450)(.06501)
$K = 4$	2.86 3.86 5.10 6.89 (.16308)(.60275)(.22666)(.00752)
$K = 5$	2.68 3.56 4.59 5.93 7.88 (.08012)(.48857)(.38191)(.04878)(.00062)
$K = 6$	2.53 3.32 4.23 5.34 6.80 8.92 (.03990)(.35788)(.46621)(.12899)(.00699)(3.9E-5)
$K = 7$	2.41 3.14 3.95 4.92 6.13 7.72 10.03 (.02030)(.24865)(.47758)(.22515)(.02761)(.00072)(1.9E-6)
$K = 8$	2.31 2.99 3.73 4.60 5.65 6.96 8.69 11.22 (.01058)(.16811)(.44044)(.31023)(.06648)(.00410)(5.6E-5)(7.6E-8)

Table 7C4. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(2.0, 1.0)$

Abscissa and (Weight)	
$K = 1$	2 (1.0)
$K = 2$	1.52 4.10 (.81530)(.18470)
$K = 3$	1.30 3.12 7.48 (.63745)(.35446)(.00809)
$K = 4$	1.18 2.67 5.72 12.96 (.51344)(.45493)(.03151)(.00012)
$K = 5$	1.10 2.40 4.88 9.93 21.76 (.42937)(.50800)(.06177)(.00085)(6.2E-7)
$K = 6$	1.04 2.23 4.39 8.48 16.72 35.85 (.37147)(.53434)(.09172)(.00245)(8.2E-6)(1.3E-9)
$K = 7$	1.00 2.11 4.07 7.63 14.30 27.61 58.30 (.33065)(.54641)(.11818)(.00472)(3.4E-5)(3.0E-8)(1.E-12)
$K = 8$.97 2.02 3.85 7.07 12.86 23.62 44.95 93.88 (.30121)(.55105)(.14032)(.00734)(8.3E-5)(1.7E-7)(4.E-11)(4.E-16)

Table 7C5. Abscissa Locations and Weights for Lognormal Distribution, $\sim\text{LN}(4.0, 1.5)$

Abscissa and (Weight)	
$K = 1$	4 (1.0)
$K = 2$	3.14 6.62 (.75371)(.24629)
$K = 3$	2.71 5.20 10.00 (.52080)(.45930)(.01990)
$K = 4$	2.44 4.47 7.88 14.45 (.36491)(.55740)(.07697)(.00072)
$K = 5$	2.25 4.02 6.77 11.42 20.36 (.26461)(.58054)(.14957)(.00527)(1.3E-5)
$K = 6$	2.12 3.70 6.07 9.82 16.13 28.19 (.19912)(.56643)(.21874)(.01554)(.00017)(1.3E-7)
$K = 7$	2.01 3.47 5.59 8.81 13.88 22.38 38.56 (.15517)(.53692)(.27632)(.03083)(.00075)(2.9E-6)(7.E-10)
$K = 8$	1.93 3.29 5.23 8.10 12.46 19.29 30.68 52.28 (.12478)(.50303)(.32096)(.04925)(.00197)(1.8E-5)(2.7E-8)(2.E-12)

CHAPTER 8.

ANALYSIS OF GAUSSIAN QUADRATURE

In Chapter Seven, we observed that Gaussian quadrature might provide high-order accuracy in the evaluation of expected costs of DDP. Because the total-cost function is roughly describable as an n -fold third-order polynomial function of the stochastic variables, it appears that coarse stochastic discretizations of $K = 2$ may be sufficient to obtain a high level of accuracy, even for arbitrary but smooth probability distributions. As a result, we may only require solutions for 2^m different outcomes of the stochastic variables, significantly reducing the exponential growth in effort compared with alternate quadrature methods.

In this chapter, we evaluate the benefit of using the Gaussian quadrature methods of Chapter Seven. The following analysis uses a range of stochastic discretizations applied in the multilinear DDP and GDP algorithms. In all cases, we use the four-reservoir test problem of Chapter Six. The four-reservoir problem contains two stochastic variables that represent independent inflows into the two uppermost reservoirs.

A. GAUSSIAN QUADRATURE ACCURACY IN ESTIMATING THE EXPECTED COST-TO-GO

This section presents quadrature error associated with stochastic discretizations from $K = 1$ (i.e., the deterministic solution) to $K = 8$. The measure of error is average absolute relative error (AARE) and absolute relative error (ARE) which we used in Chapter Six. In all cases, the state discretization is $\Lambda = 4$. For GDP, this is a sufficiently fine state discretization that interpolation error is small. For multilinear DDP, this is much too coarse for an accurate solution, but the results give some idea of the impact of Gaussian quadrature in these applications.

1. The Stochastic Models

If the condition of certainty equivalence holds, the error associated with the $K = 1$ case should be approximately zero. In Chapter Six, we modeled inflows as normally distributed $\sim N(2.0, 0.5)$ and $\sim N(4.0, 0.75)$. These normally-distributed stochastic models

produce results that are close to the condition of certainty equivalence, and accurate quadrature is less important.

As a result, we will also consider additional lognormally-distributed stochastic models that produce results that are further from the condition of certainty equivalence. The first two stochastic models are lognormally distributed with the same mean and standard deviation as the normally-distributed models (i.e., $\sim\text{LN}(2.0, 0.5)$ and $\sim\text{LN}(4.0, 0.75)$). These stochastic models are used in the "lognormal" four-reservoir problem. The second two stochastic models are also lognormally distributed, but with the variance quadrupled (i.e., $\sim\text{LN}(2.0, 1.0)$ and $\sim\text{LN}(4.0, 1.5)$). These stochastic models are used in the "high-variance lognormal" four-reservoir problem.

The abscissas and weights for the normal four-reservoir problem are calculated using Gauss-Hermite quadrature. The finest discretization used in this problem is $K = 7$ as this is the largest number of abscissa that does not produce a negative inflow (see Table 7C1: an inflow is negative if the standard deviation is $< (-) 4$ for stream #1 or $< (-) 5.33$ for stream #2). The abscissas and weights for the lognormal and high-variance lognormal problems are calculated using the Gaussian quadrature method for arbitrary distributions (Tables 7C2 through 7C5). The finest discretization used in these problems is $K = 8$.

2. Error Versus Stochastic Discretization

Tables 8A1 through 8A3 present the average absolute relative error (AARE) for the normal, lognormal, and high-variance lognormal problems. For all three problems solved using GDP, Gaussian quadrature is very accurate with a stochastic discretization of only $K = 2$. Using multilinear DDP, Gaussian quadrature is also very accurate in the normal and lognormal problems, and moderately accurate in the high-variance lognormal problem. Note that in all cases, a stochastic discretization of $K = 3$ improves quadrature accuracy only slightly, and finer discretizations produce little or no improvement.

For the normal problem, the error of the $K = 1$ solutions is modest (Table 8A1). This indicates that the normal problem is close to the condition of certainty equivalence. The lognormal problem is not significantly different, and the error of the $K = 1$ solution is not significantly greater.

In contrast, the high-variance lognormal problem, the error of the $K = 1$ solution is much more significant. High variance also increases quadrature error in all other solutions. Quadrature error is roughly five times greater for the GDP solutions and roughly forty times greater for the multilinear DDP solutions, regardless of the stochastic discretization. Larger variance produces a wider range of outcomes, and a wider range of

outcomes increases the impact of constraints and the impact of state discretization (as the outcomes sample a larger number of subdomains in the cost-to-go functions). As we will see in Section B, the inaccurate state discretization of the multilinear problem may contribute to the significantly greater increase in quadrature error.

It is noteworthy that, for multilinear DDP, the error of the $K = 1$ solution is smaller than for the GDP methods. A likely explanation for this smaller error is that the $K = 1$ solution is exact for the Gaussian quadrature when the cost function of equation (7B1) is a "first-order polynomial." In other words, Gaussian quadrature applied to equation (7B1) is exact when the cost function is linear in each dimension. In all three models, multilinear DDP uses a cost-to-go function that is closest to this form.

Table 8A1. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization (Normally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	1.2621	2.8034	2.7942
$K = 2$.0842	.0266	.0264
$K = 3$.0384	.0105	.0102
$K = 4$.0402	.0093	.0090
$K = 5$.0145	.0037	.0032
$K = 6$.0241	.0070	.0067
$K = 7$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 7$) used as estimate of exact solution for each interpolation method

Table 8A2. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization (Lognormally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	1.2945	2.8238	2.8161
$K = 2$.1044	.0255	.0212
$K = 3$.0403	.0143	.0126
$K = 4$.0328	.0080	.0074
$K = 5$.0216	.0045	.0045
$K = 6$.0187	.0079	.0073
$K = 7$.0238	.0058	.0056
$K = 8$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each interpolation method

Table 8A3. Error (% AARE) of Gaussian Quadrature with Stochastic Discretization
(High-Variance Lognormally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	7.0484	10.1995	10.1842
$K = 2$	1.6414	.1614	.1512
$K = 3$	1.1656	.0671	.0647
$K = 4$.9084	.0527	.0424
$K = 5$	1.0040	.0618	.0627
$K = 6$	1.0310	.0418	.0416
$K = 7$	1.0470	.0189	.0177
$K = 8$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each interpolation method

3. Error Bias of Gaussian Quadrature

We may gain some insight into the performance of Gaussian quadrature by observing the bias of the above errors. Tables 8A4 through 8A6 present the average relative error (ARE) for the normal, lognormal, and high-variance lognormal problems, paralleling the results in Tables 8A1 through 8A3.

In all cases, the $K = 1$ solution underestimates the exact solution. The $K = 1$ solution is a deterministic solution that does not adequately incorporate the cost of extreme outcomes. With increasing stochastic discretization, these costs are incorporated with reasonable accuracy.

Solutions with finer stochastic discretizations have less consistent biases, but some patterns are apparent. GDP solutions for the normal model show a consistent pattern of alternating positive and negative bias. If we look again at Table 7C1, we see that, when K is odd, there is a single heavily-weighted and centrally-located abscissa. In contrast, when K is even, there is a pair of heavily-weighted and centrally-located abscissa. As a result, solutions with even K have similar errors, and solutions with odd K have similar errors. Though more complex, GDP solutions for the lognormal models also consistent patterns that reflect the placement and weighting of abscissas.

In all multilinear DDP solutions for the high-variance lognormal model (Table 8A6), the biases are negative. As we will see in Section B, this may offer a clue to why the errors increase much more dramatically for multilinear DDP than for GDP.

Table 8A4. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization
(Normally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	(-) 1.2603	(-) 2.8034	(-) 2.7942
$K = 2$	(-) .0217	(-) .0190	(-) .0204
$K = 3$	(-) .0035	.0048	.0053
$K = 4$.0299	(-) .0029	(-) .0041
$K = 5$	(-) .0070	.0011	.0012
$K = 6$.0241	(-) .0014	(-) .0025
$K = 7$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 7$) used as estimate of exact solution for each interpolation method

Table 8A5. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization
(Lognormally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	(-) 1.2934	(-) 2.8238	(-) 2.8161
$K = 2$	(-) .0732	(-) .0144	(-) .0100
$K = 3$.0164	.0015	(-) .0014
$K = 4$	(-) .0107	.0012	.0026
$K = 5$.0073	(-) .0006	(-) .0006
$K = 6$	(-) .0094	.0005	.0005
$K = 7$	(-) .0173	.0002	.0012
$K = 8$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each interpolation method

Table 8A6. Error Bias (% ARE) of Gaussian Quadrature with Stochastic Discretization
(High-variance Lognormally Distributed Inflows)

Method:	Multilinear DDP	First-order GDP	Second-order GDP
$K = 1$	(-) 7.0484	(-) 10.1995	(-) 9.9922
$K = 2$	(-) 1.1774	(-) .0890	(-) .1043
$K = 3$	(-) 1.0668	(-) .0353	(-) .0175
$K = 4$	(-) .6709	.0250	.0001
$K = 5$	(-) .8847	.0535	.0322
$K = 6$	(-) .9455	.0252	.0310
$K = 7$	(-) .5859	.0012	.0149
$K = 8$	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each interpolation method

4. Comparison with a Heuristic Quadrature Method

As a brief illustration of the accuracy of Gaussian quadrature, we can compare the above results with a heuristic quadrature method. For example, *Johnson et al.* [1993] approximate lognormal distributions using abscissas at the 5, 50, and 95 percentiles of the cumulative distribution and weights of 1/6, 2/3, and 1/6.

Abscissas can be located by transformation of the appropriate standard normal deviates. The standard normal deviates can be identified from a table of the cumulative normal density frequency distribution [*Snedecor and Cochran*, 1989, p. 465]. If we define $w = \text{Ln}[s]$, then $\text{Ln}[\mu_s] = \mu_w + 0.5 \cdot \sigma_w^2$ and $\sigma_s^2 = \mu_s^2 (\text{Exp}[\sigma_w^2] - 1)$ [*Bras*, 1990]. These expressions can be rearranged to yield

$$\sigma_w^2 = \text{Ln}[\sigma_s^2 / \mu_s^2 + 1]$$

$$\mu_w = -0.5 \text{Ln}[(\sigma_s^2 / \mu_s^2 + 1) / \mu_s^2]$$

to identify the mean and standard deviation to apply to the standard normal deviates before transformation back to a lognormal variable. Table 8A7 presents the abscissas and weights of this heuristic method for the normal, lognormal and high-variance lognormal problems.

Table 8A8 presents the solution errors using the heuristic quadrature method and some previous solution errors using Gaussian quadrature. The results show that Gaussian quadrature achieves significantly greater accuracy than the heuristic method. Even using the coarsest stochastic discretization of $K = 2$ (the minimum to still produce a stochastic problem), Gaussian quadrature is over ten times more accurate. To achieve similar levels of accuracy using heuristic methods may require fine stochastic discretizations that significantly increase computational effort.

Table 8A7. Abscissa Location and Weights for Heuristic Quadrature Method

	Abscissas (*)		
<u>Normal Distributions</u>			
~N(2.0,0.5)	1.18	2	2.82
~N(4.0,.75)	2.77	4	5.23
<u>Lognormal Distributions</u>			
~LN(2.0,0.5)	1.29	1.94	2.91
~LN(4.0,.75)	2.90	3.93	5.34
<u>High-Variance Lognormal Distributions</u>			
~LN(2.0,1.0)	0.82	1.79	3.89
~LN(4.0,1.5)	2.06	3.75	6.80
(Weight)	(.16667) (.66667) (.16667)		

* Abscissas located at 5%, 50%, and 95% quantiles of each distribution. Weights are the same for all distributions.

Table 8A8. Error (% AARE) of a Heuristic Quadrature Method and Gaussian Quadrature

Quadrature method (*):	Heuristic ($K = 3$)	Gaussian ($K = 3$)	Gaussian ($K = 2$)
Normal Distribution	.2706	.0102	.0264
Lognormal Distribution	.6607	.0126	.0212
High-Variance Lognormal Distribution	2.6755	.0647	.1512

* Second-order GDP in all cases. Gaussian quadrature with $K = 7$ (normal distributions) and with $K = 8$ (lognormal distributions) used as estimate of exact solution. Error in average absolute relative error (%).

B. IMPACT OF PIECEWISE NATURE OF COST-TO-GO APPROXIMATIONS ON SOLUTION ACCURACY

In Chapter Seven, we observed that state discretization can have a variable impact on quadrature accuracy. On one hand, finer subdivision of the state space can degrade quadrature accuracy as outcomes sample more subdomains in a portion of the domain. On the other hand, finer subdivision of the state space can improve quadrature accuracy as adjacent subdomains use interpolating functions that are more accurate, consistent, and less prone to oscillation. This section assesses the impact the state discretization can have on quadrature accuracy.

1. Quadrature Error Versus Interpolation Error

Table 8B1 presents the total errors of solutions of the normal four-reservoir problem using a range of state and stochastic discretizations. This total error combines interpolation error (from coarse state discretization) and quadrature error (from coarse stochastic discretization). To evaluate the impact of state discretization on quadrature accuracy, we want to avoid the confounding influence of interpolation error.

Table 8B1. Total Error (% AARE) of Gaussian Quadrature with State Discretization (Normally Distributed Inflows and First-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	30.8370	4.2349	2.9845	2.7235
$K = 2$	30.6682	1.7480	.3119	.0268
$K = 3$	30.3765	1.6577	.2899	.0102
$K = 4$	30.5319	1.6919	.2958	.0093
$K = 5$	30.4601	1.6802	.2939	.0030
$K = 6$	30.5164	1.6793	.2943	.0070
$K = 7$	30.4536	1.6827	.2946	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each state discretization.

2. Quadrature Error with State Discretization

Table 8B2 present the errors for the same solutions as Table 8B1 but with interpolation error removed. For each state discretization, the finest stochastic discretization ($K = 7$) is used as an estimate of the exact solution.

In this case, we see that quadrature error decreases with increasing state discretization. Apparently, the increasing accuracy of interpolation overcomes the increasing fragmentation of the interpolation.

Indeed, the largest errors occur when $\Lambda = 2$ and there is only one subdomain. These suggest that deviations of the cost function from the desired polynomial form are due less to discretization of the state space than to the impact of constraints. With a coarse discretization, costs near the boundary of the state space are not very accurate, and, when constraints become binding at the boundary, the cost function (as a function of the stochastic variables) changes significantly. This is supported by the observation that, when $\Lambda = 2$, the bias of errors is mostly negative (i.e., expected costs are lower than for the finest discretization used as the "exact" solution). For coarse stochastic discretizations K , fewer outcomes sample extreme costs that occur when constraints are binding. Low expected costs estimated using coarse discretizations suggest that costs increase disproportionately when constraints are binding.

As a result, inaccurate interpolation can also degrade quadrature accuracy. This is the clue that may identify why errors increased much more dramatically for multilinear DDP than for GDP when variance was increased (Table 8A3). Because multilinear DDP is inaccurate with a coarse state discretization of $\Lambda = 4$, the form of the cost function changes significantly when constraints become binding. With higher variance, more outcomes sample extreme costs that occur when constraints are binding, and costs are poorly estimated by a polynomial function.

Tables 8B2 and 8B3 present solutions for the normal four-reservoir problem using first-order and second-order GDP. Likewise, Tables 8B4 and 8B5 present solutions for the lognormal problem, and Table 8B6 and 8B7 present solutions for the high-variance problem. As observed in Chapter Six, there is not a significant difference in the accuracy of the first-order and second-order GDP algorithms. Second derivatives are not significant for the cost function and short time horizon used by the traditional four-reservoir problem. Nevertheless, second-order GDP has consistently lower error than first-order GDP. These tables also show that error consistently decreases with finer state-discretizations, though the change is not significant after $\Lambda = 4$.

Table 8B2. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Normally Distributed Inflows and First-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	5.0831	2.9047	2.8034	2.7235
$K = 2$.1435	.0600	.0266	.0268
$K = 3$.0903	.0244	.0105	.0102
$K = 4$.0770	.0224	.0093	.0093
$K = 5$.0676	.0078	.0037	.0030
$K = 6$.0536	.0182	.0070	.0070
$K = 7$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 7$) used as estimate of exact solution for each state discretization.

Table 8B3. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Normally Distributed Inflows and Second-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	4.5443	2.8160	2.7942	2.7263
$K = 2$.1194	.0687	.0264	.0217
$K = 3$.0777	.0280	.0102	.0089
$K = 4$.0786	.0282	.0090	.0091
$K = 5$.0635	.0076	.0032	.0027
$K = 6$.0419	.0217	.0067	.0072
$K = 7$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 7$) used as estimate of exact solution for each state discretization.

Table 8B4. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Lognormally Distributed Inflows and First-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	5.2074	2.9307	2.8238	2.7393
$K = 2$.2135	.0623	.0255	.0262
$K = 3$.1567	.0275	.0143	.0118
$K = 4$.0705	.0194	.0080	.0084
$K = 5$.0309	.0099	.0045	.0042
$K = 6$.1188	.0170	.0079	.0067
$K = 7$.0946	.0149	.0058	.0055
$K = 8$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each state discretization.

Table 8B5. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (Lognormally Distributed Inflows and Second-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	4.6590	2.8426	2.8161	2.7426
$K = 2$.1551	.0654	.0212	.0187
$K = 3$.1386	.0288	.0126	.0103
$K = 4$.0601	.0243	.0074	.0076
$K = 5$.0375	.0089	.0045	.0039
$K = 6$.0890	.0165	.0073	.0064
$K = 7$.0772	.0169	.0056	.0052
$K = 8$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each state discretization.

Table 8B6. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (High-Variance Lognormally Distributed Inflows and First-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	14.2999	10.5758	10.1995	9.9888
$K = 2$.8278	.1500	.1614	.1701
$K = 3$.3882	.1021	.0671	.0546
$K = 4$.2318	.1269	.0527	.0440
$K = 5$.3028	.0971	.0618	.0627
$K = 6$.1220	.0491	.0418	.0442
$K = 7$.0576	.0176	.0189	.0185
$K = 8$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each state discretization.

Table 8B7. Quadrature Error (% AARE) of Gauss-Hermite Quadrature with State Discretization (High-Variance Lognormally Distributed Inflows and Second-Order Hermite Interpolation)

State discretization:	$\Lambda = 2$	$\Lambda = 3$	$\Lambda = 4$	$\Lambda = 5$
$K = 1$	13.6612	10.4047	10.1842	9.9922
$K = 2$.7486	.1612	.1512	.1686
$K = 3$.3420	.1145	.0647	.0484
$K = 4$.1626	.1269	.0424	.0336
$K = 5$.2389	.1089	.0627	.0489
$K = 6$.0929	.0558	.0416	.0386
$K = 7$.0532	.0225	.0177	.0179
$K = 8$	(*)	(*)	(*)	(*)

* Finest stochastic discretization ($K = 8$) used as estimate of exact solution for each state discretization.

CHAPTER 9.

CAUTION IN REAL-TIME OPERATION OF RESERVOIR SYSTEMS

Management of reservoir systems is commonly accomplished using forecasts of stream flow and of demand. However, experience and common sense indicate that using most likely forecasts may be an insufficiently cautious approach because the impacts of low probability events are not adequately incorporated into management policies. As an alternative, we can use methods that combine forecasting under uncertainty and management, producing flexible real-time management policies that achieve an appropriate level of caution. Using a four-reservoir test problem, we demonstrate this by comparing operational results of forecast-based policies with the results of policies based on stochastic dynamic programming. We observe that real-time management using forecasts results in costs that are greater on average and that are much greater for extreme events. This is true even when short-term forecasts are accurate, such as when stream flows are highly auto-correlated.

A. MOTIVATION

Periodic updates of stream-flow and demand forecasts are commonly used to arrive at management decisions in real-time, that is, as events occur. Such decisions are often based on pre-determined operating policies but sometimes they rely on deterministic optimization of future operations, as if forecasted flows were the only possible input.

However, if forecasts represent the most-likely stream flows, the potentiality of low probability events—such as extreme floods or droughts—is not adequately incorporated into the decision process. Constraints on system operation and excessive costs brought about by an extreme event may not be recognized until such an event becomes imminent and options for avoiding a catastrophe become quite limited. For example, managers of many water-supply systems hesitate to initiate rationing until shortages are imminent.

To hedge against extreme events, managers of reservoir systems may modify management policies but they have little guarantee that heuristically modified policies correctly balance their cost with the risk of extreme events. In some cases, these policies can even become too cautious and incur costs not justified by the avoided risk.

Mathematical modeling and stochastic optimization, in the context of what is known as "systems analysis," can combine forecasting and management, thereby developing management policies that are appropriately cautious. The purpose of this chapter is to illustrate the influence that extreme events have on system performance and the value of appropriately cautious management policies. In addition, this chapter demonstrates that new systems-analysis methods allow us to combine forecasting and management in relatively complex problems.

B. BACKGROUND

Operational experience indicates that forecast-based methods are insufficiently cautious. *Kitanidis and Andricevic* [1989] show that policies obtained from deterministic optimization using most likely estimates, also known as "deterministic feedback control" (DFC), perform poorly when compared with policies obtained from either "first-order approximation" or discrete dynamic programming. These are stochastic optimization methods that account for contingencies and, as a result, are more cautious. Though management policies based on most-likely forecasts may perform slightly better under average conditions, these policies may perform much worse under extreme conditions.

As a result, many agencies constrain system operations to heuristically incorporate greater caution in operations. One common approach is to adjust system operations based on performance during simulated recurrence of extreme historical events. While this approach incorporates additional caution, it may leave systems vulnerable to extreme events beyond those previously seen. For example, the severe 1976-77 drought in western North America caught many water management agencies unprepared. Consequently, the 1976-77 drought is now used frequently as a benchmark of system vulnerability [*EBMUB*, 1992]. In addition, the use of extreme historical events may produce system operations that are excessively cautious since they prepare for a specific extreme event whose exact duplication is impossible.

Unfortunately, it has been difficult to identify appropriately cautious real-time controls using available systems-analysis methods. Many existing methods applicable to complex problems require separating forecasting from management to allow application of deterministic optimization. Even those methods that combine forecasting and

management usually rely on some unique problem characteristic or other simplification. For example, *Kitanidis and Andricevic* [1989] applied first-order approximation [Kitanidis, 1987] to incorporate stochasticity using a small-perturbation approximation. This method is more cautious, but it still may not correctly balance the cost of hedging with the risk of extreme events when the uncertainty is large. Past difficulties in applying systems-analysis methods to realistic system models have contributed to limited application of these methods in practice [Rogers and Fiering, 1986].

As a result, system management has relied largely on experience. Indeed, this approach appears to have been effective in identifying an appropriate level of caution for many well-established systems [Kelman *et al.*, 1990]. However, experience may not be a sufficient guide to manage changing conditions and extreme events. For example, constraints added to trigger additional caution when extreme events become likely may produce controls that are insufficiently cautious, too cautious, or both depending on the conditions.

Discrete dynamic programming (DDP) is the most general formal optimization procedure for combining forecasting and management in reservoir control problems that meet certain requirements. It is but one of several dynamic-programming procedures available to solve dynamic control problems; however, it is the only one that is generally applicable to non-linear problems, stochastic problems, and problems where the goal is to minimize both the frequency and severity of failure. The development of new DDP methods in Chapters Five and Seven allows us to address systems that are more complex than traditionally possible.

C. SYSTEM MODELS

To compare real-time management policies using DFC and DDP, we use the four-reservoir problem (Figure 6A1) and an operating horizon of twenty-four stages ($N = 24$) stages. Streamflows are lognormally distributed $\sim \text{LN}(2.0, 1.5)$ and $\sim \text{LN}(4.0, 1.5)$. The policy using DFC is to select release decisions that minimize total cost for all remaining stages assuming a perfect forecast. At the beginning of each stage, a new forecast is generated and new release decisions evaluated. The policy using DDP is to select release decisions that minimize the sum of current and expected future costs. Expected future costs in each stage are estimated by cost functions developed by the DDP.

As seen in Chapter Eight, the traditional four-reservoir problem is close to certainty equivalent. The term "certainty equivalent" means simply that uncertainty may be neglected and decisions may be based mean values. This is due to the use of a

quadratic penalty function and also due to the small effect that constraints have on system operations. We can see this more clearly by interpreting the test problems' uncorrelated stream flows as a series of annual flow values. With this interpretation, total system storage is 800% of average annual inflow. This ratio of storage to flow is rarely achieved in practice and diminishes the influence that finite reservoir storage has on system operations. As a result, the problem is close to certainty equivalent, and DFC using the best-estimate forecast of future stream flows performs almost as well as DDP.

Because the traditional four-reservoir problem (Model A) is close to certainty equivalent, we introduce two modifications (Models B and C). These modifications result in models that are more realistic and that better illustrate the value of cautious management policies and the influence that extreme events have on system performance.

1. Four-Reservoir Model with Uncorrelated Streamflows (Model A)

As in the previous applications of the four-reservoir problem, streamflows in model A are uncorrelated. Figures 9C1 and 9C2 illustrate two different synthetic scenarios generated according to the probabilistic model. Table 9C1 identifies moments of the random variables and resulting Gaussian-quadrature abscissas and weights used calculate expectations.

We may think of the costs of the four-reservoir problem as multi-purpose penalties for not meeting water supply targets and for excessive flood releases. However, desired releases are smaller than mean inflows, and releases are maintained small as possible with little concern for the final reservoir level. We will see this effect later when observing the system's response in specific scenarios.

The use of the constraints and the quadratic cost functions of the four-reservoir problem results in a systems-analysis problem that is close to "certainty equivalent." Under these conditions, DFC using a best-estimate forecast will do nearly as well as DDP except for the effect of operating constraints.

Table 9C1. Parameters of Stochastic Variables for Uncorrelated Flow Model.

Stochastic Variable	Moments		Numerical Integration	
	Mean	Std. dev.	Abscissas	Weights
s_1	2.0	1.5	{1.5008, 6.5070}	{.9003, .0997}
s_2	4.0	1.5	{3.1426, 6.6240}	{.7537, .2463}

Figure 9C1. Uncorrelated-Flow Scenario for Models A and C (Example 1).

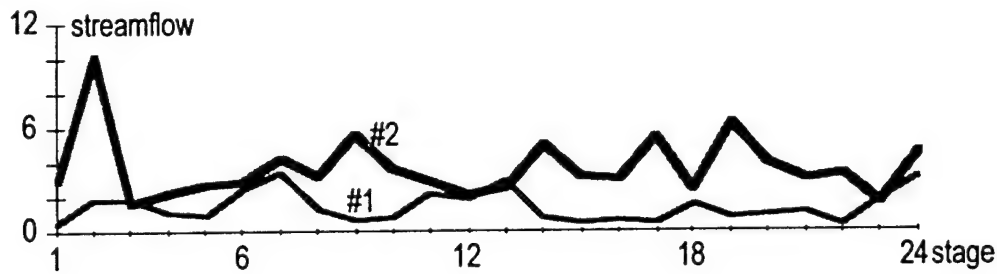
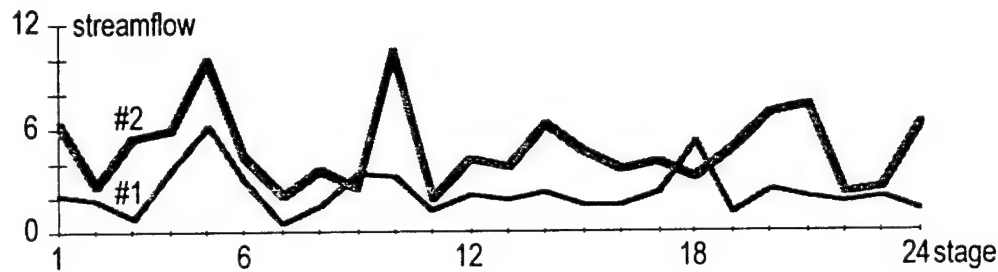


Figure 9C2. Uncorrelated-Flow Scenario for Models A and C (Example 2).



2. Four-Reservoir Model with Correlated Streamflows (Model B)

The first modification to Model A adds temporal and spatial correlation to the model of stream flows. We add auto-correlation by augmenting the traditional four-reservoir model with additional state variables, x_5 and x_6 , representing stream flows of the prior stage. Current flows are a function of these additional state variables and of stochastic variables, s (Table 9C2). Evolution of the system is now modeled by a six-state transition equation:

$$\mathbf{x}_{t+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & .8 & 0 \\ 0 & 1 & 0 & 0 & 0 & .8 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & .8 & 0 \\ 0 & 0 & 0 & 0 & 0 & .8 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{u}_t + \begin{bmatrix} 1 & 0 \\ .5 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ .5 & 1 \end{bmatrix} \mathbf{s}_t$$

where state variables are reservoir levels and prior stream flows. Figures 9C3 and 9C4 illustrate two different synthetic stream-flow scenarios generated according to this probabilistic model of inflows. Moments of the random variables have been chosen to produce stream-flow moments that match those of the uncorrelated flow model. These random-variable moments are identified in Table 9C3. Because of the highly skewed distributions of these random variables, Gaussian-quadrature produces large abscissas and small weights. These values can introduce significant solution errors. To reduce these errors, the tails of the unbounded log-normal distributions have been truncated to

produce smaller abscissas without significantly degrading expected-value calculations (Chapter Seven).

With stream-flow correlation, constraints on reservoir capacity have a greater effect and result in a model that is further from the condition of certainty equivalence. We can see this more clearly by interpreting the problem's correlated stream flows as a series of monthly flow values. With this interpretation, the model covers a 2-year period of operations and total system storage is 67% of average annual inflow. This is more realistic, and as we will see, the results of this model better illustrate the value of cautious management policies.

Table 9C2. Model of Correlated Stream Flows.

Stream Flow:	Mean	σ	Model
#1	2.0	1.5	$0.8x_5 + s_1$
#2	4.0	1.5	$0.8x_6 + 0.5s_1 + s_2$

Table 9C3. Parameters of Stochastic Variables for Correlated Flow Model.

Stochastic Variable	Moments		Max. Tail		Numerical Integration	
	Mean	σ	Value	Prob.	Abscissas	Weights
s_1	0.4	0.9	25.0	.000088	{.3019, 7.4922}	{.9867, .0132}
s_2	0.6	0.6364	25.0	<.000001	{.4583, 3.4516}	{.9527, .0473}

Figure 9C3. Correlated-Flow Scenario for Model 2 (Example 1).

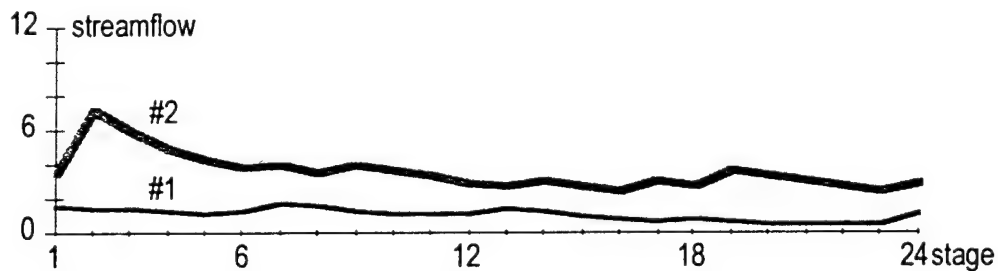
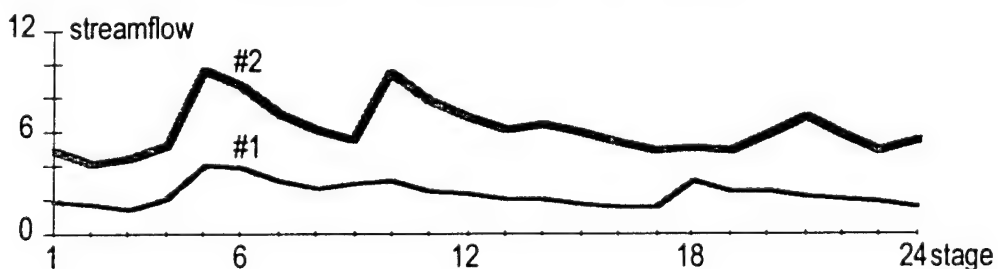


Figure 9C4. Correlated-Flow Scenario for Model 2 (Example 2).



3. Four-Reservoir Model with Higher-Order Cost Function (Model C)

The second modification to Model A changes the penalty functions from quadratic to the fourth-order:

$$F(\mathbf{x}) = \sum_{j=1}^4 (x_j - b_j)^4$$

$$C(\mathbf{u}) = \sum_{k=1}^4 a_k(u_k - 1.0)^4$$

These non-quadratic penalties result in a model that is also further from certainty equivalent. Compared with results using Models A and B, results using Model C most dramatically display the poor performance of the DFC policy.

D. RESULTS

For each model, we compared real-time performance of the DFC policy with performance of the DDP policy using a thousand equally-likely scenarios generated according to the probabilistic model of inflows. In addition, we illustrate specific system behavior under each policy by viewing representative scenarios. Each scenario consists of synthetic flow series of 24 values for each of the two streams. State and stochastic discretizations are sufficient to reduce error below 1% AARE.

Table 9D1 provides a quick summary of the mean, minimum, and maximum of costs for the two models under each management policy. For each model, we can see that operating costs under the DFC policy are generally greater because the policy is insufficiently cautious when compared with the appropriately cautious DDP policy.

Table 9D1. Mean, Minimum, and Maximum of Cost Distributions.

Flow Model	Policy	Mean	Median	Minimum	Maximum
A (traditional)	DFC	1173.3	1143.6	618.0	3817.2
	DDP	1122.2	1097.3	648.4	3139.2
B (correlated)	DFC	1474.5	1149.3	260.6	96773.5
	DDP	1361.8	1056.9	369.9	89299.4
C (non-quadratic)	DFC	26743.0	19018.3	5805.9	2368884.5
	DDP	20257.8	17030.8	7105.6	1103842.5

1. Results for the Uncorrelated Flow Model (A)

Applying the DFC policy and DDP policy to the thousand scenarios of the uncorrelated-flow model, we can compare the policies' performance. Figure 9D1a shows the distribution of costs resulting from DFC policy using best-estimate forecasts and Figure 9D1b shows the distribution of costs resulting from DDP policy. By subtracting the resulting cost of the DDP policy from the resulting cost of the DFC policy in each scenario, we can better compare performance of the two policies. Figure 9D1c shows the distribution of these cost differences for the thousand scenarios, with negative values indicating better performance for the DFC policy and positive values indicating better performance for DDP policy.

Neither policy performs consistently better than the other; however, on average, the cautious DDP policy performs better than the DFC policy. DFC does well in scenarios with low flow variability, and does better than DDP in 20.1% of the scenarios. On the other hand, the caution of DDP does well in scenarios with high flow variability, and DDP does better in 79.9% of the scenarios. More telling is the policies' relative performance in extreme scenarios. In the best scenario, the DFC policy outperforms the DDP policy by achieving a cost that is 7% (i.e., 45.5 penalty units) lower. Note that, relative to the DDP policy, the best scenario for the DFC policy is not the scenario with the lowest cost but the scenario that most closely matches the real-time forecasts. In contrast, in the worst scenario, the DFC policy under-performs the DDP policy and the resulting cost is 22% (i.e., 678.0 penalty units) higher.

Release decisions applied to specific scenarios can be observed to better assess the forecast-based DFC policy. Figures 9C1 and 9C2 illustrate "favorable" and "unfavorable" synthetic streamflow scenarios in the sense that the resulting costs as measured by the penalty functions are lower and higher than average. These are not the extreme scenarios discussed above, but more typical examples selected from among the first ten out of the thousand.

The first scenario presents an outcome for which the DFC policy performs better than the DDP policy. Figures 9D2a-d show, for each reservoir respectively, the release decisions and evolution of storage levels from application of the DFC policy. For comparison, the figures also show as dashed lines the releases and storage levels resulting from application of the appropriately cautious DDP policy. DFC decisions maintain a relatively constant release level throughout the twenty-four stages. Because penalties associated with releases are greater than potential penalties associated with the ending storage, the DFC policy tends toward the filling of reservoirs. On the other hand, DDP

releases are unnecessarily cautious for this scenario: higher releases are maintained until the last few stages. During these last few stages, the DDP policy reduces releases as it is unlikely that flows during the remaining stages will require the large available storage as a buffer to spread out releases of large inflows. Performance costs are low under both policies, and the DFC policy outperforms the DDP policy by 4% (i.e., 795.4 to 828.2 penalty units).

The second scenario presents an outcome for which the DFC policy performs worse than the DDP policy. Figures 9D1a-d show, for each reservoir respectively, the release decisions and evolution of storage levels. In this scenario, the forecast-based DFC decisions are insufficiently cautious: reservoirs fill rapidly and less buffering capacity is maintained to spread out release of later flows. Resulting costs are high, and the DFC policy underperforms by 10% (i.e., 1952.1 to 1775.2 penalty units).

Figure 9D1a. Distribution of Costs for DFC Policy (Model A).

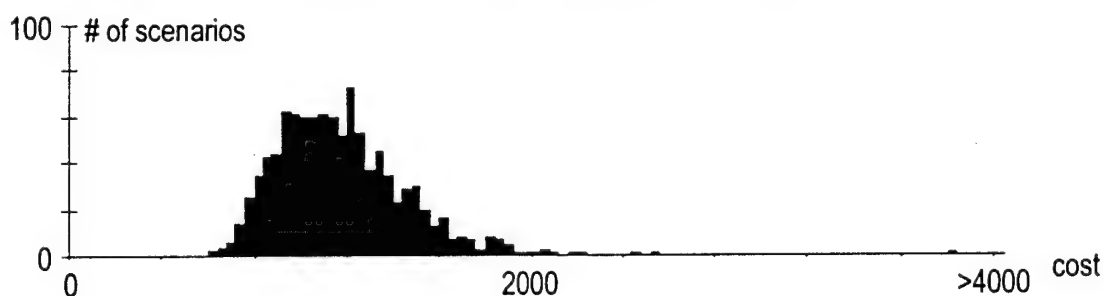


Figure 9D1b. Distribution of Costs for DDP Policy (Model A).

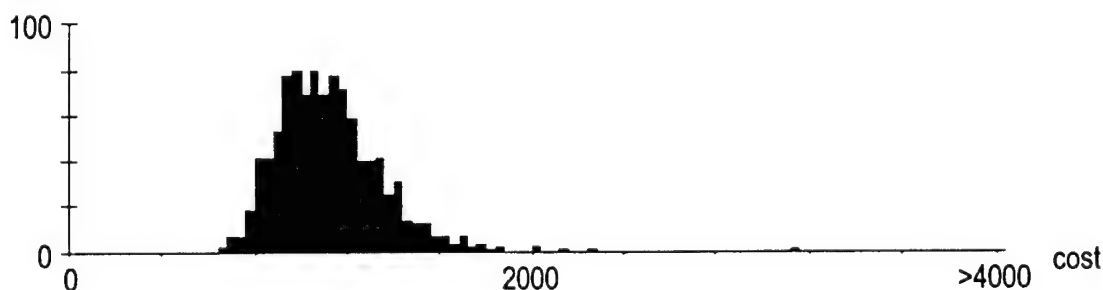


Figure 9D1c. Distribution of Cost Differences for DFC Versus DDP (Model A).

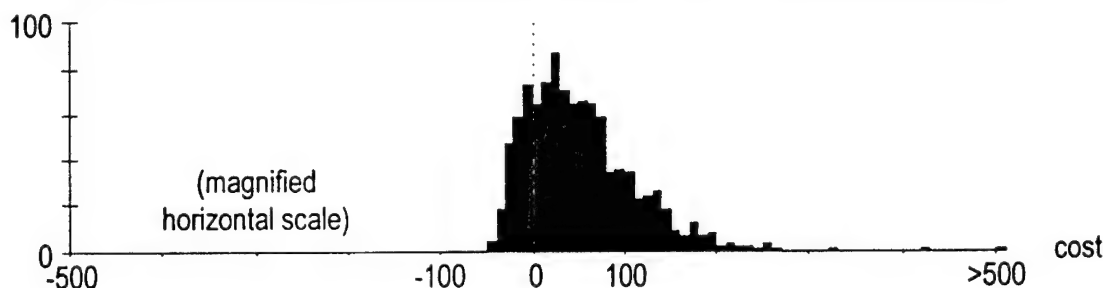


Figure 9D2a. Release and Storage for 1st Reservoir (Model A, Example 1).

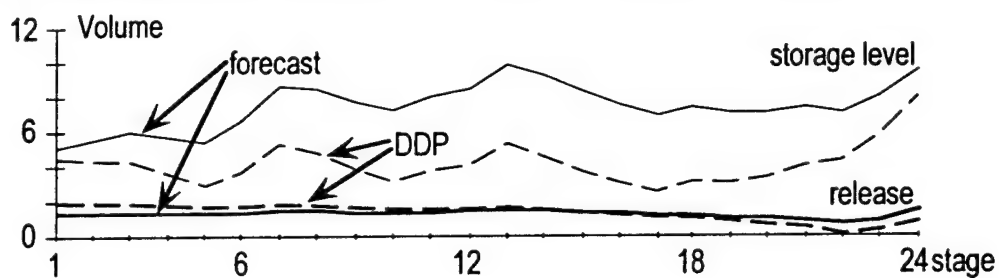


Figure 9D2b. Release and Storage for 2nd Reservoir (Model A, Example 1).

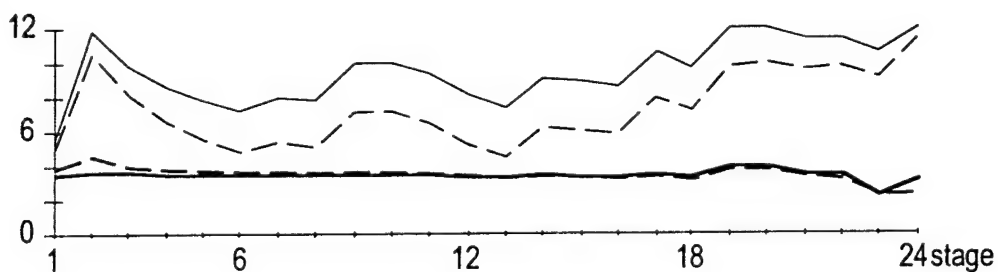


Figure 9D2c. Release and Storage for 3rd Reservoir (Model A, Example 1).

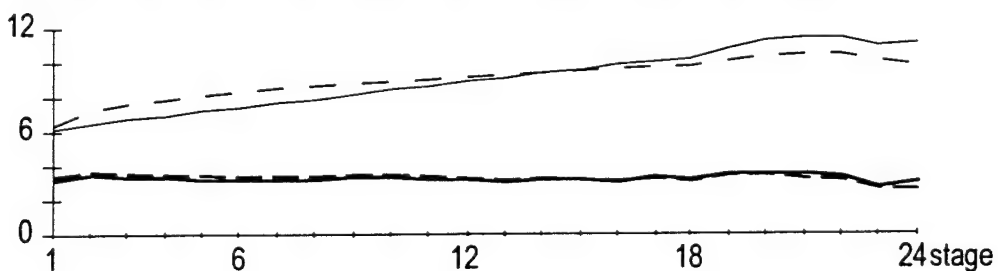


Figure 9D2d. Release and Storage for 4th Reservoir (Model A, Example 1).

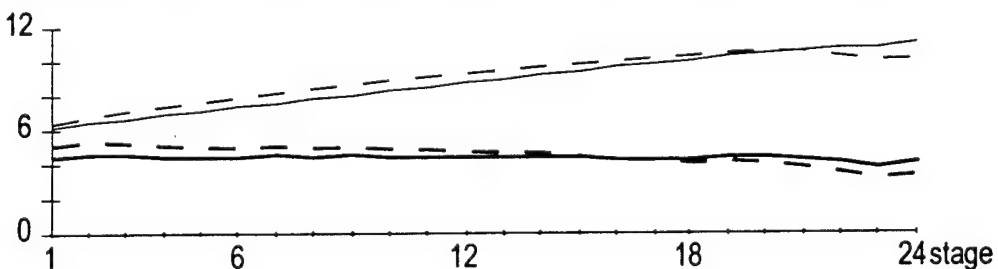


Figure 9D3a. Release and Storage for 1st Reservoir (Model A, Example 2).

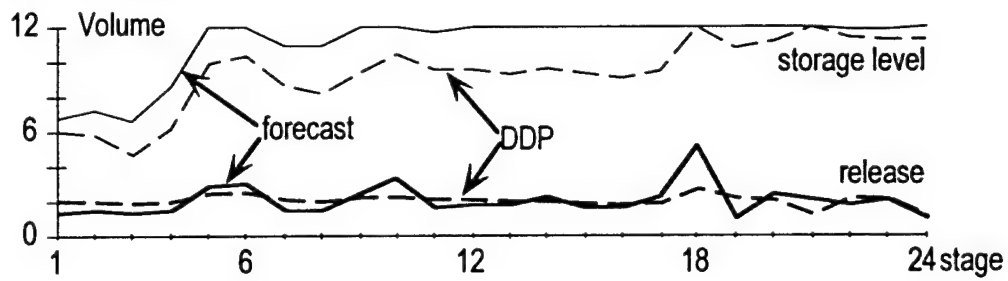


Figure 9D3b. Release and Storage for 2nd Reservoir (Model A, Example 2).

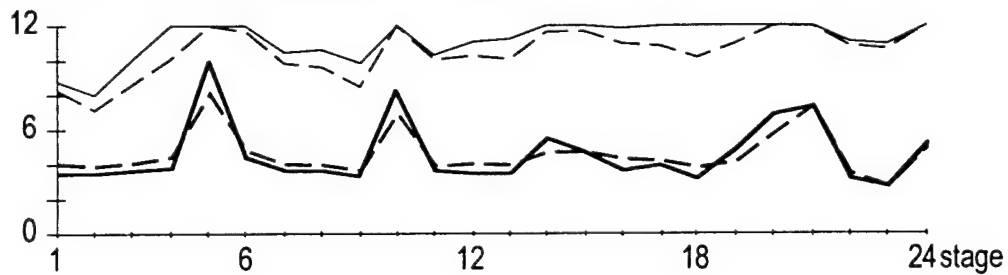


Figure 9D3c. Release and Storage for 3rd Reservoir (Model A, Example 2).

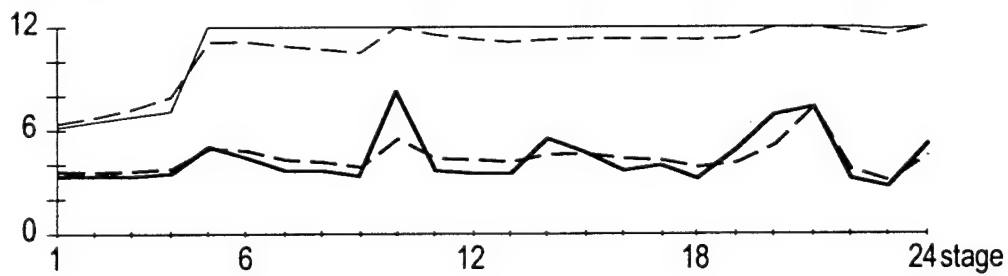
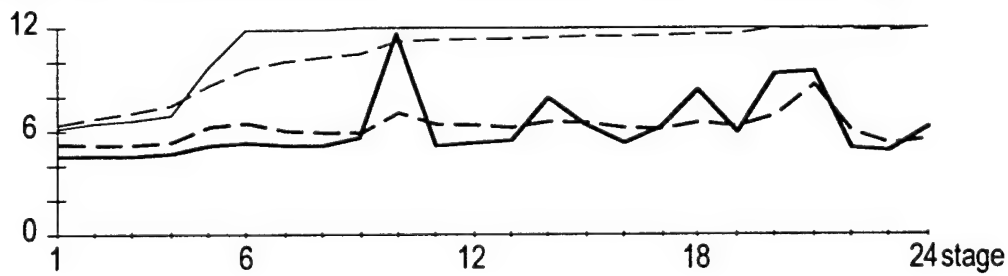


Figure 9D3d. Release and Storage for 4th Reservoir (Model A, Example 2).



2. Results for the Correlated Flow Model (B)

Figures 9D4a-c show system performance using the correlated-flow model. These figures parallel Figures 9D1a-c for the traditional uncorrelated-flow model. Average cost for model B is greater than for model A, even though median costs are roughly equal. This is largely because the cost of extreme events is much greater for model B. The coincidence and persistence of flows resulting from cross-correlation and auto-correlation causes extreme high combined flows and causes high flows to persist longer, producing higher costs. This also causes extreme low flows to persist, producing lower costs (Table 9D1). The DFC policy does better than the DDP policy in only 17.8% of the scenarios. In the best performing scenario, the DFC policy outperforms the DDP policy by achieving a cost that is 13% (i.e., 137.4 penalty units) lower. However, in the worst performing scenario, the DFC policy underperforms by 8% with a substantial difference in cost of 7474.1 penalty units.

Figures 9C3 and 9C4 illustrate two synthetic flow scenarios for the correlated-flow model. In contrast to the uncorrelated-flow scenarios in Figures 9C1 and 9C2, we can see that the correlated synthetic flows change less rapidly and high flows show greater persistence. The correlated and uncorrelated scenarios of use the same seed in a random number generator (i.e., they have the same position among the thousand scenarios) so that differences in the streamflow models can be compared directly.

As before, the first scenario presents an outcome for which the DFC policy performs better than the DDP policy. Figures 9D5a-d show, for each reservoir respectively, the release decisions and evolution of storage levels. As in the uncorrelated-flow model, the DFC policy maintains relatively constant release levels throughout the twenty-four stages. Again, releases under the DDP policy prepare the system for high inflows that do not happen, and high releases are maintained until the last few stages. Costs are low under both policies, and the DFC policy outperforms the DDP policy by 11% (i.e., 656.0 to 741.1 penalty units).

The second scenario presents an outcome for which the DFC policy performs worse than the DDP policy. Figures 9D6a-d show, for each reservoir respectively, the release decisions and evolution of storage levels. In this scenario, the DFC policy has failed to anticipate high flows that cause the reservoirs to fill rapidly. With the DDP policy, reservoir filling is delayed, though available storage capacity is still lost early. Resulting costs are high, and the DFC policy underperforms by 6% (i.e., 3056.4 to 2877.5 penalty units).

Figure 9D4a. Distribution of Costs for DFC Policy (Model B).

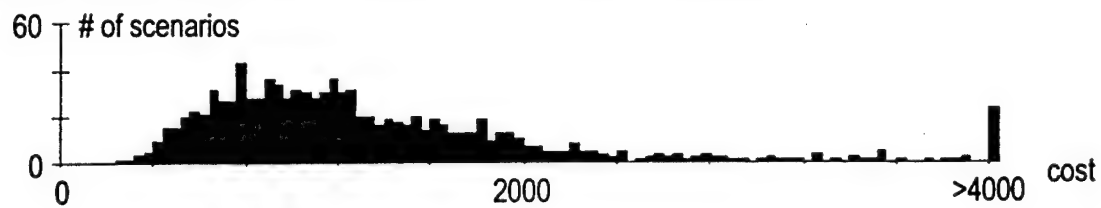


Figure 9D4b. Distribution of Costs for DDP Policy (Model B).

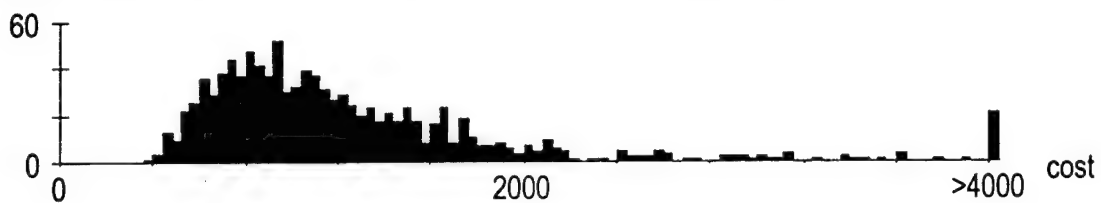


Figure 9D4c. Distribution of Cost Differences for DFC Versus DDP (Model B).

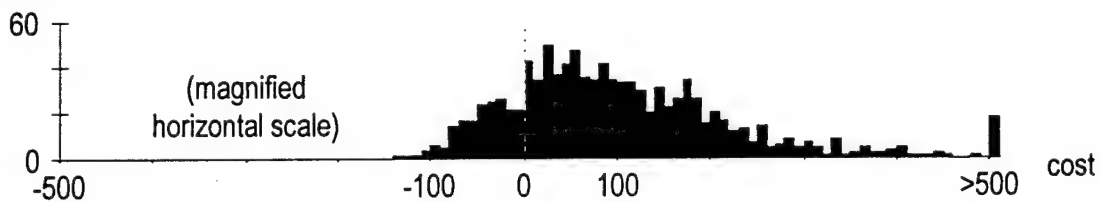


Figure 9D5a. Release and Storage for 1st Reservoir (Model B, Example 1).

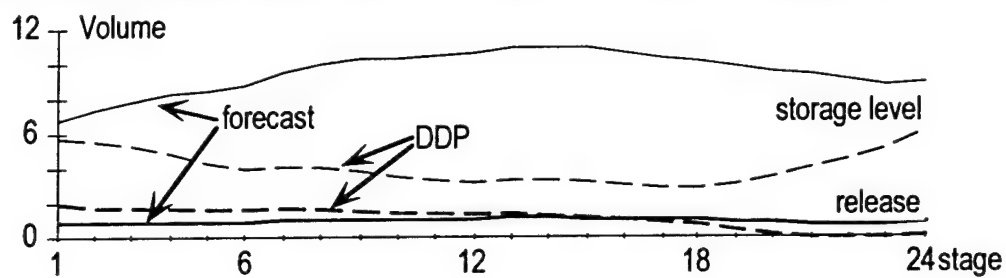


Figure 9D5b. Release and Storage for 2nd Reservoir (Model B, Example 1).

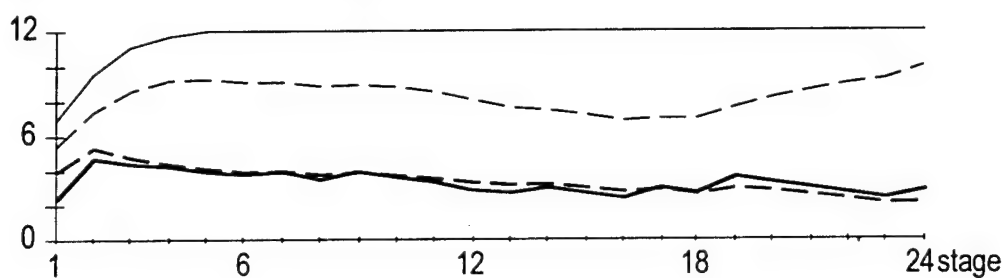


Figure 9D5c. Release and Storage for 3rd Reservoir (Model B, Example 1).

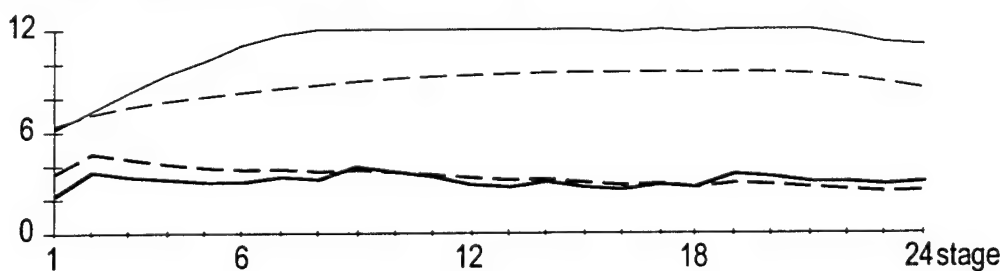


Figure 9D5d. Release and Storage for 4th Reservoir (Model B, Example 1).

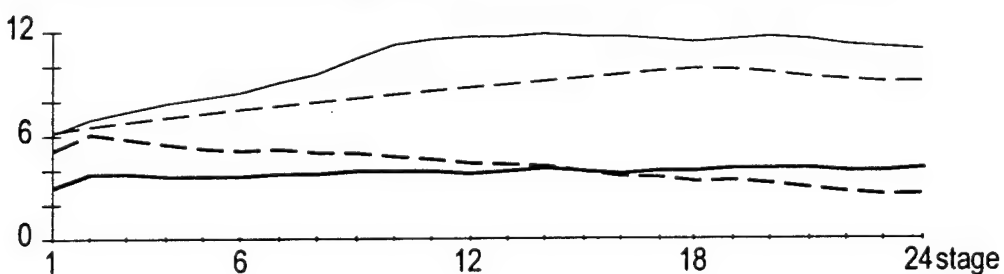


Figure 9D6a. Release and Storage for 1st Reservoir (Model B, Example 2).

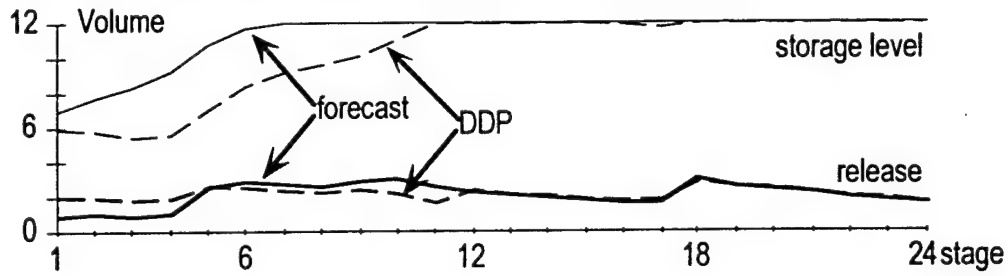


Figure 9D6b. Release and Storage for 2nd Reservoir (Model B, Example 2).

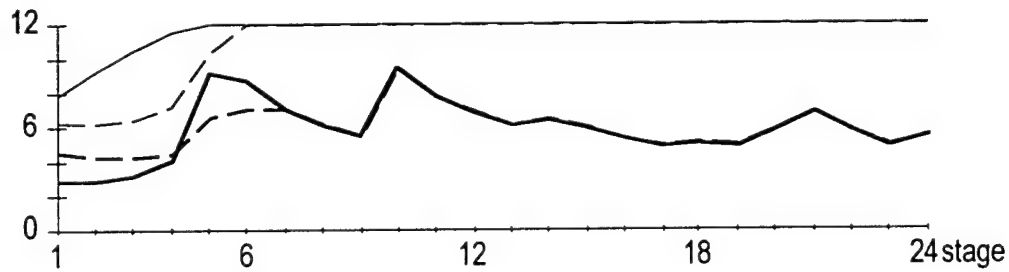


Figure 9D6c. Release and Storage for 3rd Reservoir (Model B, Example 2).

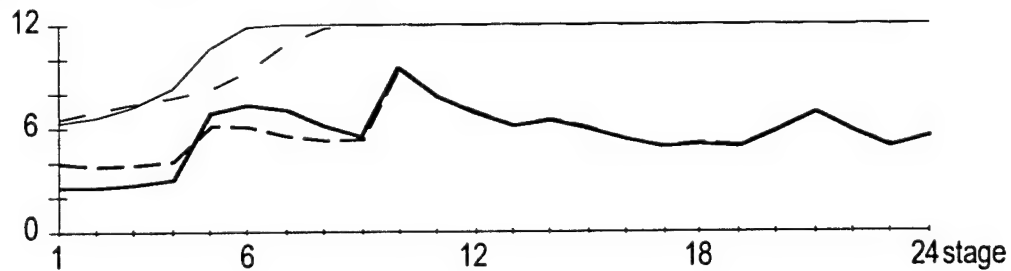
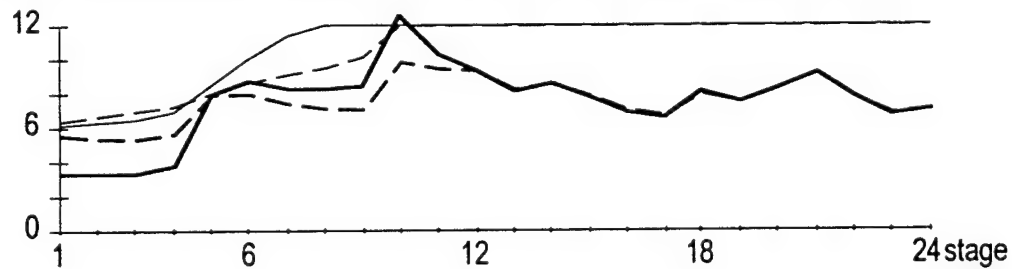


Figure 9D6d. Release and Storage for 4th Reservoir (Model B, Example 2).



3. Results for the Non-Quadratic Penalty Model (C)

Figures 9D7a-c show system performance using the non-quadratic cost function. These figures parallel Figures 9D1a-c and Figures 9D4a-c. Compared to Models A and B, costs for Model C are greater because of the fourth-order penalty. Using this model, the DFC policy does better than the DDP policy in 23.9% of the scenarios, and the poor performance of the DFC policy is dramatic. In the best performing scenario, the DFC policy outperforms the DDP policy by achieving a cost that is 22% (i.e., 2285.8 penalty units) lower. However, in the worst scenario, the DFC policy underperforms and the cost is 110% (i.e., 1.2 million penalty units) higher!

Figures 9C1 and 9C2 illustrate two synthetic flow scenarios for this model. These are the same scenarios used in Model A. Again, the first scenario presents an outcome for which the DFC policy performs better than the DDP policy. Figures 9D8a-d show, for each reservoir respectively, the release decisions and evolution of storage levels for the favorable first scenario. As seen in Models A and B, the DFC policy maintains relatively constant release levels and the DDP policy dictates additional releases that are unnecessary in hindsight. The DFC policy outperforms the DDP policy by 15% (i.e., 8877.5 to 10404.2 penalty units).

The second scenario presents an outcome for which the DFC policy performs worse than the DDP policy. Figures 9D9a-d show, for each reservoir respectively, the release decisions and evolution of storage levels. In this scenario, the forecast-based DFC policy allows the reservoirs to fill rapidly. In contrast, buffering capacity is maintained under the DDP policy. Under both policies, release decisions are higher than in model A, though they are not significantly different otherwise. Resulting costs, however, are significantly different: the DFC policy underperforms by 53% (i.e., 67205.4 to 43965.2 penalty units).

Figure 9D7a. Distribution of Costs for DFC Policy (Model C).

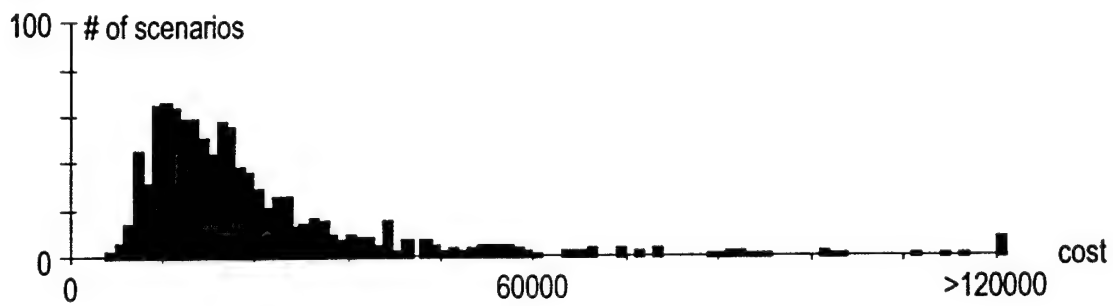


Figure 9D7b. Distribution of Costs for DDP Policy (Model C).

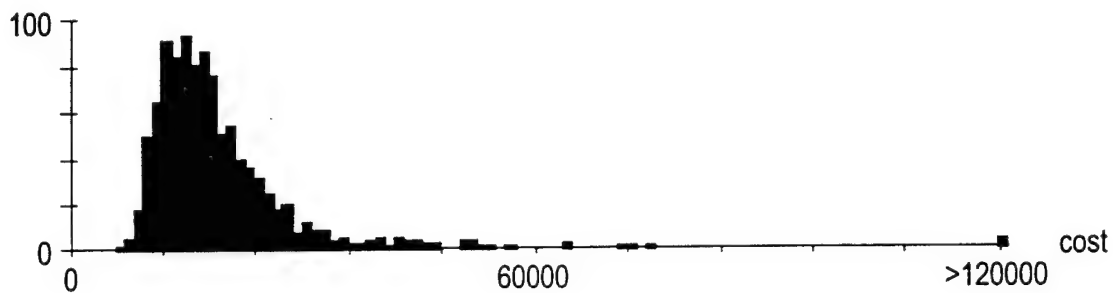


Figure 9D7c. Distribution of Cost Differences for DFC Versus DDP (Model C).

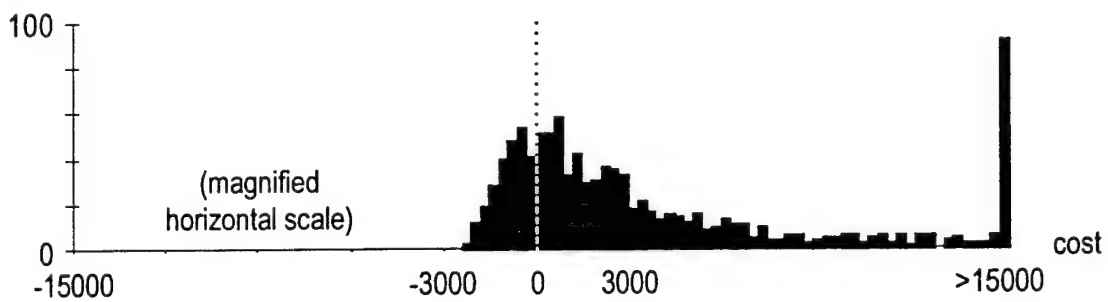


Figure 9D8a. Release and Storage for 1st Reservoir (Model C, Example 1).

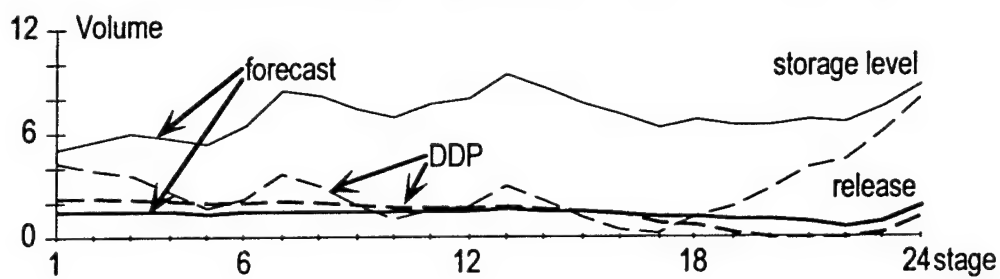


Figure 9D8b. Release and Storage for 2nd Reservoir (Model C, Example 1).

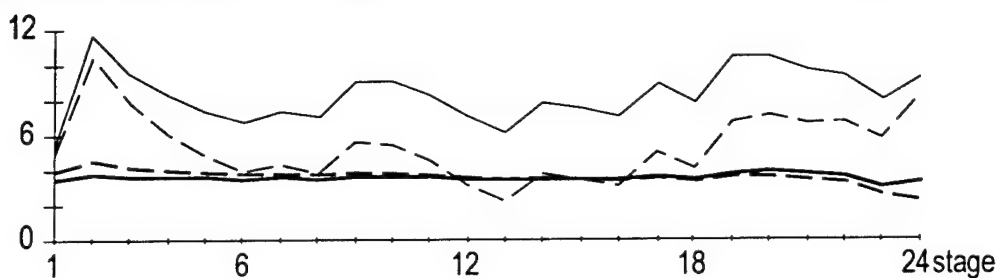


Figure 9D8c. Release and Storage for 3rd Reservoir (Model C, Example 1).

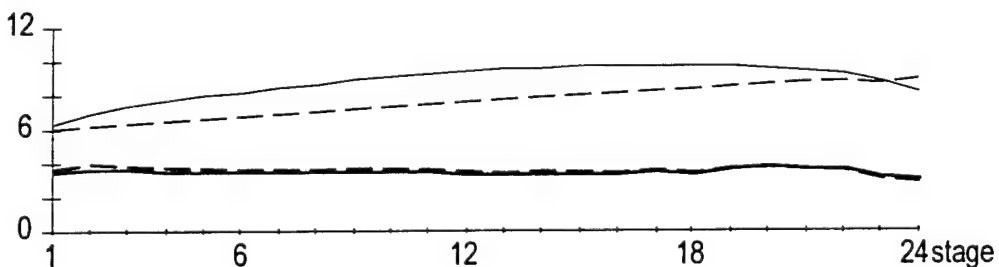


Figure 9D8d. Release and Storage for 4th Reservoir (Model C, Example 1).

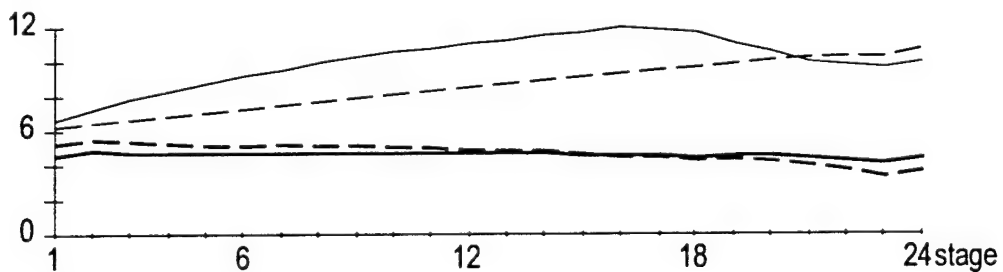


Figure 9D9a. Release and Storage for 1st Reservoir (Model C, Example 2).

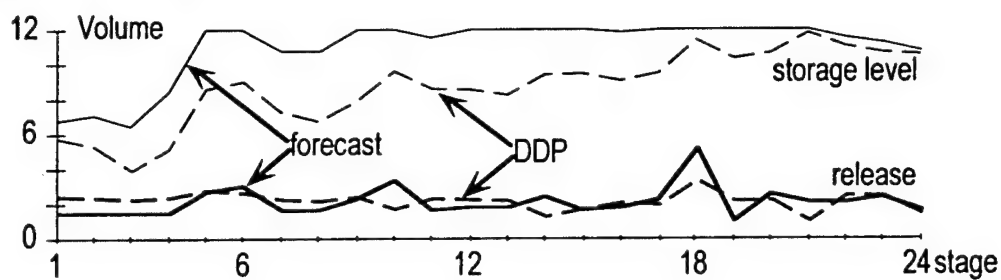


Figure 9D9b. Release and Storage for 2nd Reservoir (Model C, Example 2).

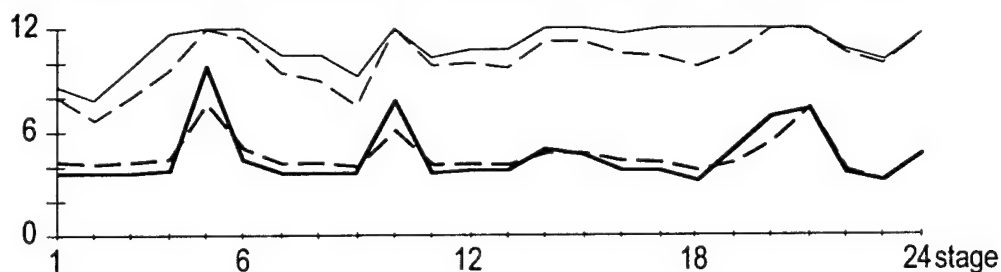


Figure 9D9c. Release and Storage for 3rd Reservoir (Model C, Example 2).

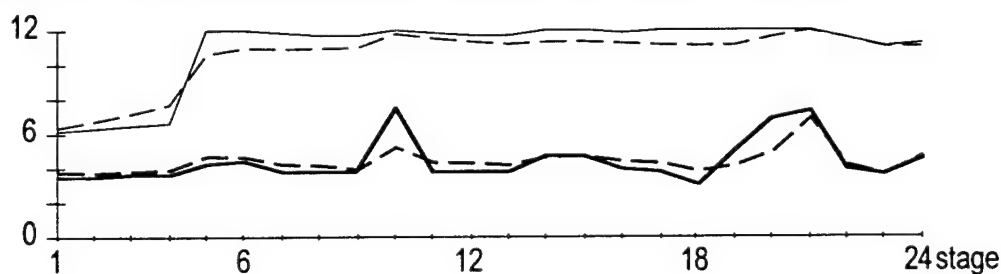
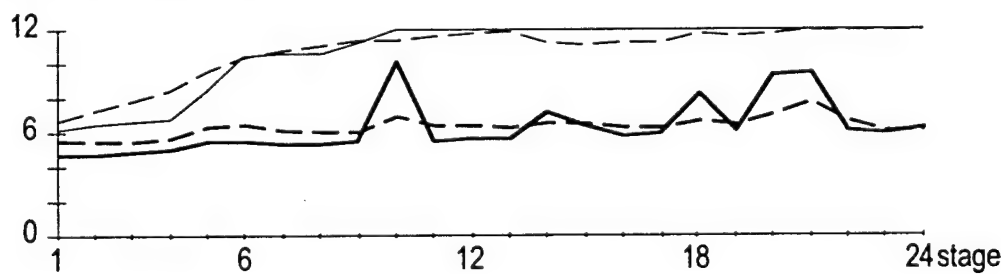


Figure 9D9d. Release and Storage for 4th Reservoir (Model C, Example 2).



E. CONCLUDING REMARKS ON THE CAUTIOUS MANAGEMENT OF RESERVOIR SYSTEMS

Efficient reservoir-system operation is difficult when driven by uncertain inputs such as stream flow. Because of limitations on practical application of existing operations analysis methods, system managers have relied on forecast-based DFC policies as the best available method to conduct real-time operations. Because it is recognized that DFC policies are insufficiently cautious, heuristic constraints on system operations have frequently been adopted. However, these provide no guarantee of achieving an appropriate level of caution.

The results presented in this chapter demonstrate the poor performance of DFC. Simulated real-time operational results for DFC policies have been compared with DDP policies in a variety of problems. The first uses the well-known four-reservoir model. Because this model is close to certainty equivalent, we have introduced two additional models that are further from certainty equivalent. The first includes temporal and spatial correlation. The second uses a non-quadratic cost function that heavily penalizes deviations from target release and reservoir levels. Application of DFC and DDP to these problems demonstrate the under-performance of DFC on average and the extremely poor performance possible in extreme events. These results also demonstrate that, even if more accurate forecasts are available, the information can be used with other methods such as DDP to produce a better policy.

Except in problems that are appropriately modeled as certainty equivalent, forecast-based policies are insufficiently cautious and generally result in poor performance. Certainty equivalence occurs when system performance is measured by a quadratic cost function and there are no binding constraints on operations. However, realistic systems are not certainty equivalent. Except for systems characterized by extremely large storage capacities and by quadratic cost functions (perhaps valid for some hydropower systems), system performance will be significantly degraded by application of forecast-based policies.

The modifications used to develop Models B and C are only a couple of those that may be needed to represent many real systems with practical accuracy. Many systems are even more severely constrained. Reservoir capacities may be even more severely limited, or additional constraints may be required to incorporate various legal requirements and social values. Seasonal variation in inflows will also increase the effect of constraints on system operation. Many systems cannot be modeled using a quadratic cost function. For example, many systems cannot tolerate the complete emptying of a

reservoir or the reduction of water supplies to zero, and these conditions would imply an infinite cost. We anticipate that for many real systems, Model C represents more accurately than Model A the importance of appropriately cautious policies.

Examples presented here also demonstrate the feasibility of applying DDP to problems of greater complexity than generally assumed. We apply DDP to 24-stage problems to develop the first published solution of a six-state variable DDP problem. This has allowed us to combine forecasting under uncertainty and management in systems analysis applied to reasonably complex multi-reservoir problems.

CHAPTER 10.

VALUATION OF WATER RESOURCES

To identify shortage costs in a value model, we need to develop an appropriate cost function. Some reservoir management purposes, such as power generation, have clearly defined cost or benefit functions that can be quantified by revenue generated or costs incurred. We accept these revenues and costs in defining cost functions because they have been determined by market mechanisms. Other reservoir management purposes have less clearly defined costs because market prices may not be available or easily quantified. For example, the cost of water shortage is not well defined because water prices are not usually defined by the market.

The purpose of this chapter is to propose and apply a surrogate cost function for shortages. This cost function allows us to explicitly state (and critique) the cost of rationing given by the general equation

$$\text{Cost of rationing} = \frac{\alpha}{1+\alpha} P_0 [Q_0 - Q (Q/Q_0)^{1/\alpha}]$$

where P_0 is a market price for an available quantity of water Q_0 , and α is the "elasticity" of water demand. This equation assumes that market prices provide an appropriate measure of the benefit of water use and that the elasticity of demand is constant.

By developing such a cost function, water management agencies can explicitly quantify the expected impact of water rationing on consumers. What is more important, agencies can use such a cost function to evaluate the expected benefits and costs of different operating policies and planning scenarios. As a result, different options for system expansion or modification can be compared with greater objectivity.

A. BACKGROUND ON MARKET PRICES

We cannot easily quantify water supply benefits without a water market. Instead, we can simulate a market mechanism to evaluate the benefit of water use and the cost of water shortage. In this section, we will first consider how real markets identify prices.

1. An Ideal Market

When the price of a commodity is free to change, it responds to market forces of demand (i.e., desire for a commodity) and supply (i.e., willingness to sell). We generally observe that as the price goes up, we buy less of the commodity (with the exception of some odd commodities, called “superior goods,” whose high price may create demand, e.g., jewelry). We also observe that as the price goes up, producers of the commodity are willing to supply more. Figure 10A1 illustrates the impact of price on the demand and supply of a typical commodity. For some commodities however, other possibilities also exist. For example, a commodity may have production costs that decrease with quantity (e.g., commodities produced with economies of scale), or the quantity of a commodity may not respond to price (e.g., commodities that cannot be produced or reproduced).

Note that there is an equilibrium unit price P^* for each quantity Q that matches demand with supply. P^* identifies the “market clearing” price of a commodity. If the price is too high ($> P^*$), then demand is less than the supply (P_1 and Q_1 in Figure 10A1). In this case, the price drops as producers compete to sell their supply of the commodity. If prices are set too low ($< P^*$), then demand is greater than the supply (P_2 and Q_2 in Figure 10A1). In this case, the price will be driven up as consumers compete for limited supplies. If the price is not allowed to change, then rationing is required to allocate available quantities of the commodity, perhaps through regulation by a government agency. When supply and demand are in balance, the market clearing price efficiently allocates the available quantities without regulation.

The supply and demand functions are surrogates for the benefits of consumption and the costs of production. The demand function identifies a price that estimates the benefit from consuming each additional unit of the commodity when the total supply is at level Q . If the benefit is less, then we consume less; if the benefit is more, then we consume more. The net benefit to consumers is the difference between the price they pay for the commodity and the benefit they achieve from its use. Economists define this benefit as the “consumer surplus” (Figure 10A2). Likewise, the supply function identifies a price that estimates the cost of producing each additional unit of the commodity when total production is at level Q . The net benefit to producers is the difference between the cost to produce the commodity and the price they can charge. Economists define this benefit as the “producer surplus” (Figure 10A2).

The market clearing price associated with the equilibrium between demand and supply has the benefit of maximizing the combined consumer and producer surpluses (i.e., the net benefit). For example, if a government applies controls to lower consumer

prices, then producers may respond by reducing production. Even though this may result in a larger consumer surplus, this comes at the expense of the producer surplus (Figure 10A3). As a result, there is a net loss in total benefits to society. Also, demand Q_D is greater than supply Q (since the benefit of consumption P_D is greater than the price P), and rationing is required to allocate available quantities of the commodity.

Figure 10A1. Example Demand and Supply Functions

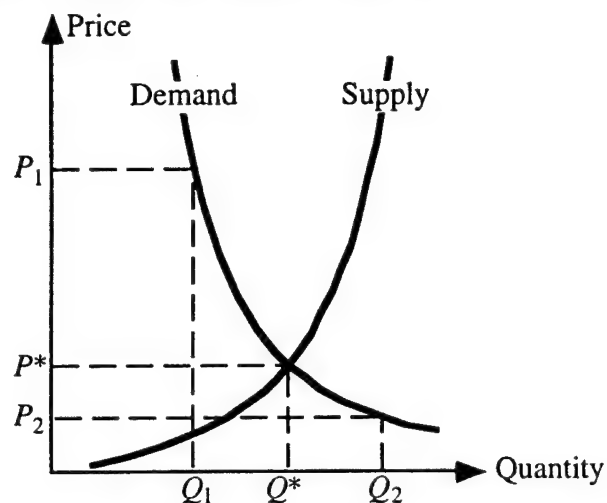


Figure 10A2. Consumer and Producer Surplus

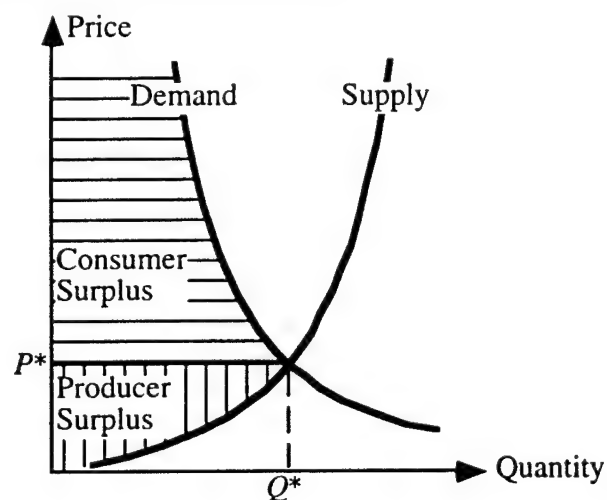
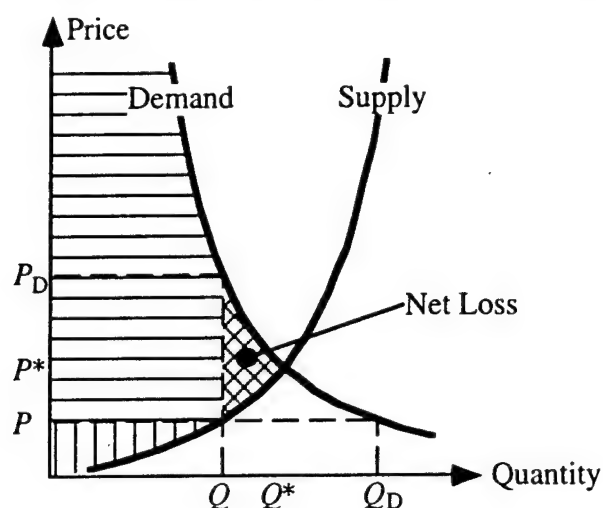


Figure 10A3. Impact of Non-Market Price on Consumer and Producer Surplus



2. Non-Ideal Water Markets

In contrast to the ideal market presented above, the supply of water does not usually change in response to market forces. Instead, the supply of water is constrained by hydrologic conditions that are not affected by price. Figure 10A4 illustrates the relationship between price and quantity when hydrologic conditions constrain supply. This supply function also shows that an agency or other entity may establish a threshold price required to provide any supply.

Likewise, the demand for water does not usually change in response to market forces. Because water utilities are natural monopolies (i.e., sole source suppliers) they have considerable power in setting prices. As a result, water supplies are typically government owned or subject to government regulation [Zarnikau, 1994], and utilities that manage these supplies are often non-profit or regulated-profit agencies. Thus, the price for water is often based on some measure of capital and operating costs and not on the cost of scarcity [Moncur, 1989; Rosa, 1991]. Because demand for water is growing almost universally, prices are often below market clearing prices. These low prices are popularly justified by the essential needs that water meets in human consumption, food production, and industry. However, without a market to allocate supplies, agencies must regulate who receives a portion of the limited supply (i.e., agencies must ration, even under “normal” water supply conditions).

As long as the price is below the market clearing price for the available supply, the actual price serves only to divide water-use benefits between the utility and consumers. Under such conditions, an agency could establish a higher price as a form of

taxation without decreasing the benefit of water use. More likely, such taxation would improve the efficient allocation of water supplies as the actual price is brought closer to the market clearing price. Figure 10A4 illustrates the division of this benefit in terms of producer and consumer surpluses. We may interpret the threshold supply price as a "production" cost that we subtract from the producer surplus. In the short term, this may include the costs of operating storage, transportation and treatment facilities. Over the long term, this may also include the capital costs for these facilities.

Nevertheless, a market relationship between price and demand exists even if the price is not allowed to change in response to market forces. As a result, we can use a simulated market price to estimate the benefit for each unit of water. The total benefit for consuming Q units is given by

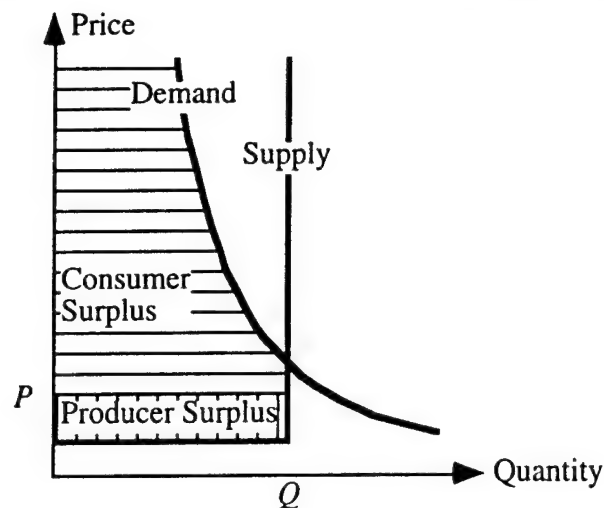
$$\text{Total benefit} = \int_0^Q P(Q) dQ \quad (9A1)$$

where P is given by the demand function as the price that consumers are willing to pay for supply Q . To estimate the net benefit (i.e., consumer and producer surpluses), we subtract "production" costs associated with providing water. To neglect these costs, we consider the price raw-water (i.e., prior to treatment, delivery, and storage) to evaluate the benefits of water use and to develop a demand function. Without these costs, the cost to "produce" water is zero until we exhaust natural supplies.

Even when prices are not allowed to change in response to market forces in the short term, the market equilibrium between demand and supply may influence prices in the long term. Over time, demands and water supply conditions change; and plans for system expansion or modification should be based on estimates of the costs incurred and benefits achieved. When the price of water is based on operating and capital costs, system expansion or modification may push the price higher. Given enough time, system expansion may increase the price until a market equilibrium is achieved.

In the short term however, the market equilibrium between demand and supply may have little influence. For example, if an agency does not adjust prices in response to limited drought supplies, then the gap will increase between the actual price and the market clearing price, making efficient allocation of supplies even less likely. Moreover, if the market price is increasing due to long-term increases in demand or decreases in supply, then there may exist a persistent gap between the actual price and the market clearing price.

Figure 10A4. Producer Surplus and Consumer Surplus for Water Supply



B. ASSUMPTIONS

To measure the cost of rationing, we need to identify what benefit we derive from water use and what loss we incur from reductions in use. We can estimate the benefit of water using the market clearing price discussed above. Under non-market conditions, this is a conservative estimate: non-market mechanisms used to allocate supplies in times of shortage may not identify the best allocation for limited supplies. As a result, the benefit from use of these supplies will be less and the cost of rationing will be more [Mercer and Morgan, 1989].

However, water markets are not common and we generally cannot identify market clearing prices directly. Instead, we identify price as a “willingness to pay” [Dandy, 1992] given the supply conditions. We can estimate this price using a reasonable demand function based on indirect observations and common sense. For example, our willingness to pay for water increases as water becomes scarce. Therefore, we should expect the price to increase monotonically as the severity of rationing increases.

Because water is essential, we may not have much flexibility in reducing consumption during times of shortage, even with the incentive of higher prices. We say that demand for water is “inelastic” with changes in price because we find it difficult or expensive to adjust to shortages. Demand elasticity is the change in the quantity demanded as a result of changes in price. Note that we can define demand elasticity even if we do not use price to influence demand.

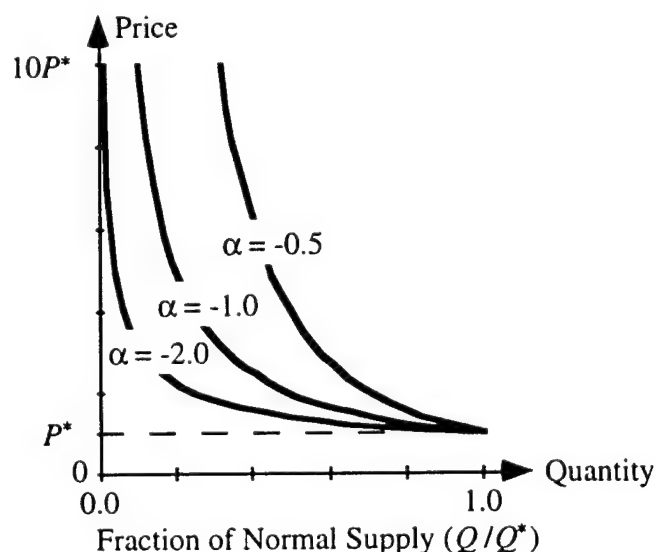
To simulate the behavior of market prices and estimate the willingness to pay, we can model demand using a reasonable mathematical function. In particular, we use the mathematical definition of demand elasticity. Demand elasticity is defined as the fractional change in demand with a small fractional change in price. Elasticities typically have negative values since demand usually decreases with increasing price [Schlette and Kemp, 1991]. Mathematically, we express the elasticity α as a function

$$\alpha = \lim_{\Delta P \rightarrow 0} \left\{ \frac{\Delta Q / Q}{\Delta P / P} \right\} = \frac{dQ / Q}{dP / P} \quad (9B1)$$

where Q is the quantity demanded and P is the price [Hanke, 1980]. For example, if the price of water changes 1%, then the change in demand for water is $\alpha\%$. The closer elasticity is to zero, the larger the change in price required to produce a change in demand (Figure 10B1). We say that demand is “inelastic” when $-1 < \alpha < 0$. In other words, demand is inelastic when the resulting fractional decrease in demand is less than a fractional increase in price.

Because we do not have market prices available to identify the benefit of water use and the cost of shortages, we estimate market prices from available data. This data includes estimates of demand elasticity and willingness to pay for water. Because this data does not cover the full range of water supply conditions, we extrapolate using simplifying assumptions. This section summarizes and justifies the use of these assumptions. Even though the resulting cost function is approximate, we can judge the appropriateness of extrapolated shortage costs since they are explicit.

Figure 10B1. Elastic ($\alpha = -2$), Isoelastic ($\alpha = -1$), and Inelastic ($\alpha = -0.5$) Demand Functions



1. Constant Elasticity

The most important assumption that we use is that elasticity is constant. A principal and practical reason for this assumption is that evaluating elasticities is difficult even for a single average value. Fortunately, water demand is consistently inelastic (even though it may vary in degree) so a reasonable value should produce reasonable results. Also, there is some evidence to suggest that, at least for urban demands, the long-term elasticity is roughly constant over a large range of prices [*Martin and Thomas, 1986*].

Moreover, constant elasticity produces a demand function that is consistent with a common-sense interpretation of water supply values (Figure 10B1). Using a constant inelastic value for α , the price of water decreases with an increasing supply and approaches zero as supplies become infinite (or, if we consider the impact of flooding, the price could become negative). Also, consistent with our inability to survive without some water, the price becomes infinite as supplies approach zero. Because water supplies are essential to the existence of life, an infinite price is appropriate. As a result, the assumption of constant elasticity produces a cost function that is more appropriate than the quadratic cost functions used by other authors [*Bogle and O'Sullivan, 1979; Foster and Beattie, 1979; Johnson et al., 1993; Kitanidis and Andricevic, 1989; Zarnikau, 1994*]. A quadratic cost function implies that the price of water is finite, even as supplies approach zero. Note also that a quadratic cost function implies that demand varies linearly with price.

In practical situations, supplies should never approach zero and produce infinite costs. Most water supply systems contain reservoirs, and these allow agencies to save water for later use when the value (i.e., willingness to pay) is higher. In these situations, the supply curve may have a positive slope for all prices (Figure 10B2). With the ability to store water, agencies should never allow complete emptying of all reservoirs if management is sufficiently cautious. Additionally, we can identify alternate supplies (e.g., trucking in water in the short term; desalination in the longer term) that we use when supplies are scarce and prices sufficiently high. In these situations, the supply curve of Figure 10A4 is not uniformly vertical, but has a positive slope when prices are high (P_{alt} in Figure 10B3).

Figure 10B2. Impact of Storage on Market Equilibrium Scenarios

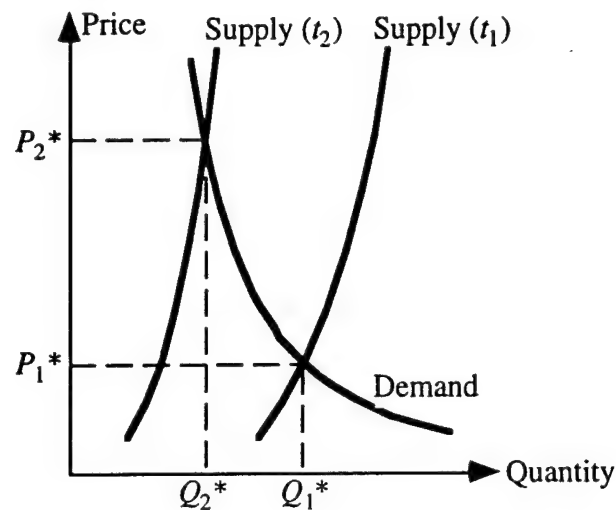
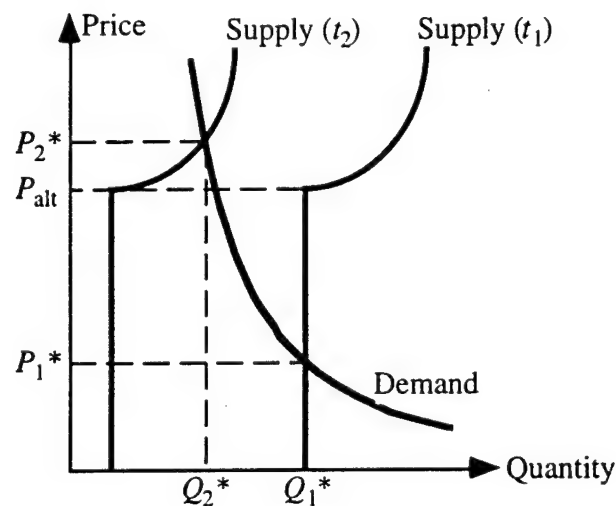


Figure 10B3. Impact of an Alternate Supply on Market Equilibrium Scenarios



2. Availability of Water Supplies

The supply function that we use to describe the availability of water does not affect our estimate of a demand function or the cost of water rationing. Figure 10A4 illustrates a potential fixed-quantity supply function that represents the amount of water available if there are no alternative supplies and there is no storage. In such situations, available supplies cannot change with price because they are controlled by hydrologic conditions. Comparing Figure 10A4 with Figures 10B2 and 10B3, we see different supply functions. Rationing has the same impact on a consumers' welfare whether it is the unavoidable result of shortage or the result of management decisions to hedge supply decisions. Though these different supply functions produce different outcomes of

equilibrium price and available quantity, the demand-function relationship between price and quantity does not change for consumers.

Over the long-term however, the impact of water rationing may depend on the supply function. For example, a supply function associated with an ongoing condition of scarcity should encourage conservation by water users and agencies (e.g., planting yards with drought tolerant plants, installing low-flow toilets, regulating the type and amount of water use), and this would change the cost of water rationing in the short-term.

3. Effect of Timing on Rationing Costs

The cost of rationing depends on the amount of time that the agency and consumers have to adjust. The amount of time available depends both on our ability to forecast shortages and on how rationing is phased in. If very little time is available, we have little flexibility in changing our water use and we incur high costs. With sufficient time or with ongoing conditions of scarcity, we may change our patterns of consumption and/or change the water-supply system to improve its resiliency and reliability. In general, the more time available, the better we can plan for shortages.

The availability of reservoirs makes a significant impact on the timing of shortages. Obviously, when we are able to store water for later use, we reduce the likelihood of shortages. In addition, we can hedge against the impact of severe shortages by initiating rationing earlier than required by available supplies. This can provide the agency and consumers with more time to adjust: low storage levels provide a strong warning of impending shortage and early rationing allows consumers to adjust water use more gradually.

As a result, the timing of shortages significantly affects the elasticity of water demand. The more time available to adjust to shortages, the greater the elasticity of demand (i.e., α becomes more negative). In developing the rationing cost function, we neither make assumptions regarding the timing of shortages nor allow for the impact that timing has on changing elasticity. It is possible to incorporate this impact in the state-space model of a system by modeling the impact of prior rationing decisions on current rationing costs. However, given our difficulty in identifying a single average elasticity value, leaving this value as a constant may still be reasonable.

4. Externalities

Market prices should be used cautiously to measure for the value of water use. Where market prices exist for a commodity, the convenience of these prices sometimes

results in their use without regard to other impacts and values not reflected in the price. We call these impacts and values “externalities” because they exist outside the market.

Potential externalities include indirect community benefits that are not reflected in the decisions of a consumer. For example, the price or reliability of water supplies may cause industrial users to change or move plant operations without including in their decisions the indirect impacts on jobs and taxes. In addition, prices do not have the same impact on all consumers. In such cases, reductions in water use may depend more on an ability to pay and less on the value of the use (although, actual prices seen by consumers are usually based on block rates that preserve a certain base level of service for a low cost). Given these non-ideal conditions, we may need to include other criteria to identify efficient prices and allocations.

C. RATIONING COST FUNCTION

Given the appropriateness of the assumptions discussed above, we can identify a demand function and a rationing cost function. To identify a demand function, we rearrange and integrate equation (9B1) to solve price as a function of demand (assuming constant elasticity α):

$$\int \frac{dP}{P} = \int \alpha^{-1} \frac{dQ}{Q}$$

$$\ln(P) = \alpha^{-1} \ln(Q) + c_1$$

$$P = c_1 Q^{\alpha^{-1}}$$

c_1 is the combined constant of integration for the indefinite integrals. Suppose “normal” supply in a period is Q_0 (perhaps established by system delivery capacity) and corresponds to an estimated market price of P_0 . We can use these values to solve for the coefficient c_1 :

$$c_1 = P_0 Q_0^{\alpha^{-1}}$$

this gives us our general form of the demand function:

$$P = P_0 (Q/Q_0)^{\alpha^{-1}} \quad (9C1)$$

assuming a constant demand elasticity α and an estimated market equilibrium at point (P_0, Q_0) . This general demand function is similar to functions developed by other authors [Dandy, 1992; Mercer and Morgan, 1989].

Now that we have a general demand function, we can identify a rationing cost function. Rationing cost is the lost benefit for each unit of water that is unavailable for consumption, either as a result of unavoidable shortages or as a result of decisions to hedge by saving water for later use. The total benefit of consuming Q units of water is given by equation (9A1) and is the area under the demand curve (Figure 10C1). However, using the price function given by equation (9C1), this value is infinite for $-1 \leq \alpha \leq 0$ (i.e., whenever demand is inelastic). Since our goal is to estimate the cost of rationing rather than the total benefit of water use, we can evaluate this cost as the difference between the benefit of supplying water at level Q and the benefit of supplying water at the "normal" level Q_0 where $Q < Q_0$. Thus,

$$\text{Cost of rationing} = (\text{benefit of } Q_0) - (\text{benefit of } Q)$$

This establishes the rationing cost of a normal supply Q_0 as zero. Thus, we can evaluate the cost of rationing as:

$$\text{Cost of rationing} = \int_Q^{Q_0} P(Q) dQ \quad (9C2)$$

Substituting equation (9C1) into equation (9C2) and integrating, we get the result presented by Dandy [1992]:

$$\text{Cost of rationing} = \frac{\alpha}{1+\alpha} P_0 [Q_0 - Q (Q/Q_0)^{1/\alpha}] , \quad \alpha \neq -1 \quad (9C3a)$$

$$\text{Cost of rationing} = P_0 Q_0 \ln(Q) , \quad \alpha = -1 \quad (9C3b)$$

The price given by equation (9C1) is the marginal cost of supplying one additional unit of water given by the negative slope of the rationing cost function. In Figure 10C2 for example, the price P_0 is the negative slope of the tangent line at $Q = Q_0$.

Figure 10C1. Graphical Estimation of the Total Benefit of Water Consumption

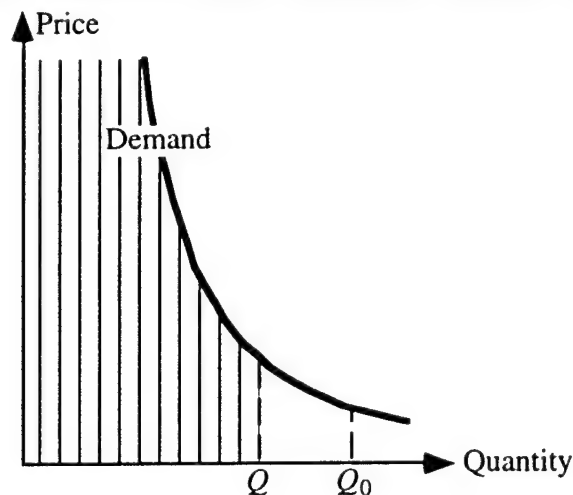
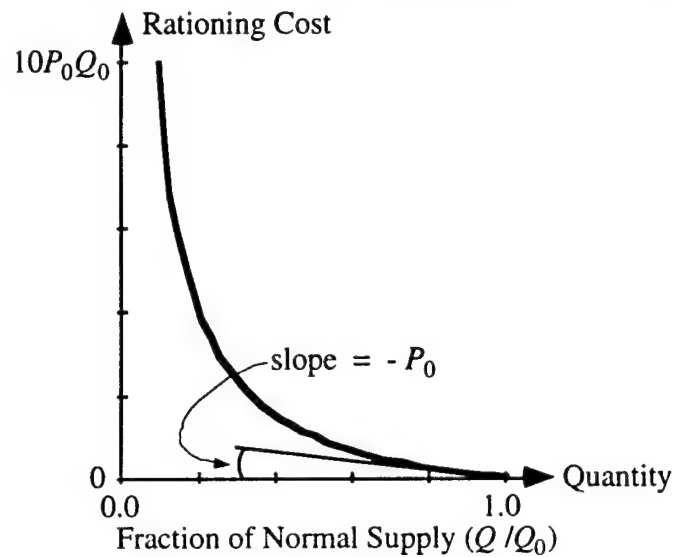


Figure 10C2. Rationing Cost Versus Fraction of Normal Supply for $\alpha = -0.5$



D. APPLICATION TO AN EXAMPLE SYSTEM

We can apply the general cost function of equation (9C3a) to develop a reasonable estimate of rationing costs for real systems. Again, this equation assumes that demand elasticity is constant over the anticipated range of water supply conditions. To identify the cost function, we need an appropriate reference point for market equilibrium, and we need an appropriate estimate of demand elasticity. In this section, we will identify reasonable parameters using an example water-supply system. We conclude this chapter by applying the resulting cost function to the example system to assess the impact of water rationing.

1. The Example Water supply System

To illustrate the application and results of the rationing cost function, we consider a hypothetical water supply system that annually supplies 600,000 acre-feet of water (roughly the water supply required for a residential district of 3,000,000 people, or about 1,200,000 families). Assuming a non-profit charter for this agency, the price of water to consumers primarily reflects a pro-rated share of the storage, transportation, and treatment costs. As a result, the agency finds it difficult to identify the market value of its water supplies and the impact of rationing decisions. The agency needs an estimate of these benefits and costs to minimize the impact of rationing and to better plan for system expansion.

This hypothetical system presents water supply conditions that are similar to expected conditions (within a few decades) for the East Bay Municipal Utilities District (EBMUD) of Oakland, California. Except for local runoff in its district, EBMUD currently obtains its entire supply from the Mokelumne River, located in the Sierra Nevada Mountains. EBMUD is the largest user and manages the two largest reservoirs on the River. Though the current demand by EBMUD is just over 200,000 acre-feet annually, the effective demand that the district must meet is greater. The River also provides water to meet growing demands by a variety of other users and for in-stream flow requirements. Because EBMUD's water rights are junior to those of most other users, it must manage the system to meet demands of all users to ensure that it can meet demands in its own district.

2. Water Prices

In normal years, wholesale water costs range from \$44 per acre-foot in San Joaquin Valley to \$237 per acre-foot in southern California [McClurg, 1992b]. During recent mild droughts in the early 1990s, California sold water from its Water Bank for \$140-\$175 per acre-foot [Howitt, 1994; McClurg, 1992a]. The more severe drought in 1976 and 1977 prompted some California communities (including Santa Barbara, Goleta and Montecito) to build desalination plants for backup supplies that cost \$1500 to \$2000 per acre-foot. At a level of 25% rationing, various authors anticipate rationing costs due to "welfare losses" to be between \$40 and \$180 per acre foot [Fisher *et al.*, 1995].

Water rates in the EBMUD district indicate that water has a high value for consumers. In recent years, the industrial water rate for EBMUD has been between \$431 and \$1,307 per acre-foot scaled on prior usage [CUWA, 1991, pp. 5.6-7]. EBMUD's ongoing Water Supply Management Program has considered a variety of efforts to encourage conservation with costs from about \$1000 to \$12000 per acre-foot to increase system yield and reliability [EBMUD, 1992, volume I, p 9.8; Fisher *et al.*, 1995]. However, water rates indicate the value of water consumption, and these rates are net of raw water costs, storage, transportation, and treatment. It is likely that the net benefit (i.e., the consumer and producer surplus) is less, even if water rates are set below market prices.

Water shortage impacts on industry can be especially severe: *California Urban Water Agencies* [1991] found significant production and employment reductions from water rationing. In some industries, the value of water can exceed \$400,000 per acre-foot. Combined with low demand elasticity, rationing costs can be astoundingly high.

Though it is likely that these industrial uses have priority over other uses when rationing is required, exceptionally severe rationing can result in high costs.

3. Demand Elasticity

While it is recognized that water demand is inelastic with changes in price, the exact value of α is not precisely known and it can be influenced by a variety of confounding conditions [Foster and Beattie, 1979; Gallager and Robinson, 1977; Headley, 1963; Moncur, 1987; Schlette and Kemp, 1991]. For example, Martin [1991] uses abundant data from the city of Tucson, Arizona, to arrive at the conclusion that α is somewhere in the range of -0.70 to -0.26, depending on conditions such as weather and local wealth.

Other studies have not gathered sufficient data to identify such a range of elasticity; however, they have still found that water demand is quite inelastic, particularly when water is scarce. Hanke [1980] estimated that elasticities are between -0.22 and -0.16 for Perth, Australia during the dry summer months. Other authors have found the demand for water to be inelastic with values typically between -0.7 and -0.2 [Billings and Agthe, 1980; Danielson, 1979; Mercer and Morgan, 1989; Moncur, 1987; Moncur, 1989].

The available time for adjustment to shortage conditions can significantly influence demand elasticity, as discussed earlier. With increasing time, elasticity should increase (i.e., become more negative). Martin [1986] compared water demands in a variety of consistently wealthy urban areas and found that demand has a long-term elasticity of about $\alpha = -0.5$. When these urban areas have less time to adjust to scarcity, demand is more inelastic [Cameron and Wright, 1990; Mercer and Morgan, 1989]. Moncur [1989] studied the drought response of the urban area of Honolulu, HI and observed a "short-run" elasticity of -0.265 and a "long-run" elasticity of -0.345. In addition, it appears that demands may become more inelastic over time [Young, 1973], possibly reflecting the impact of more intensive conservation efforts.

4. Assessment of Rationing Costs

Using a demand elasticity of $\alpha = -0.5$ and a "normal" price (negative marginal cost) of \$200 per acre-foot, we can obtain a specific rationing cost function from equation (9C3). Assuming $\alpha \neq -1$, this cost of rationing is

$$\text{Cost of rationing} = 200 Q_0 [(Q/Q_0)^{-1} - 1]$$

Cost is in dollars and demand is in acre-feet. Note that the price of water (i.e., marginal cost) associated with each level of demand is the slope of the cost function given by equation (9C1). This price is

$$P = 200 (Q/Q_0)^{-2} = -\frac{dC}{dQ}$$

where C = Cost of rationing. Applying these functions to extreme levels of rationing, we observe that 50% rationing results in a price of \$800 per acre-foot, and 90% rationing results in a price of \$20,000 per acre-foot. Rationing cost (i.e., the lost benefit of water consumption) is $\$200Q_0$ at 50% rationing and $\$1800Q_0$ at 90% rationing.

Water rationing costs calculated by assuming $\alpha = -0.5$ appear conservative considering many authors' estimates of short-term elasticity. Also, the prices of equation (9C1) and costs of equation (9C3a) appear low when compared to observed price and cost data. As noted earlier, the elasticity of $\alpha = -0.5$ estimated by *Martin* [1986] was more appropriate for urban areas that had ample time to adjust to long-term conditions of water-scarcity.

As a result, short-term elasticity should be even less negative. Using a demand elasticity $\alpha = -0.33$, the cost of rationing is

$$\text{Cost of rationing} = 100 Q_0 [(Q/Q_0)^{-2} - 1]$$

The corresponding function for the price of water is

$$P = 200 (Q/Q_0)^{-3}$$

In this case, 50% rationing results in a price of \$1600 per acre-foot (close to the cost of desalination), and 90% rationing results in a price of \$200,000 per acre-foot (about \$0.60 per gallon, close to the cost of bottled water). The rationing cost is now $\$300Q_0$ at 50% rationing and $\$9900Q_0$ at 90% rationing.

5. Application to the Example System

To place these results in context, we can estimate the impact of shortage on the hypothetical water supply system. If we assume that families in the district have similar demands for water—in other words, they have similar uses for water, derive similar benefits, and have a similar ability to pay for this water or to adjust to shortages—then we can evaluate the impact of water shortages both on a single family (with representative water demands) and on the district (as an aggregate of all users).

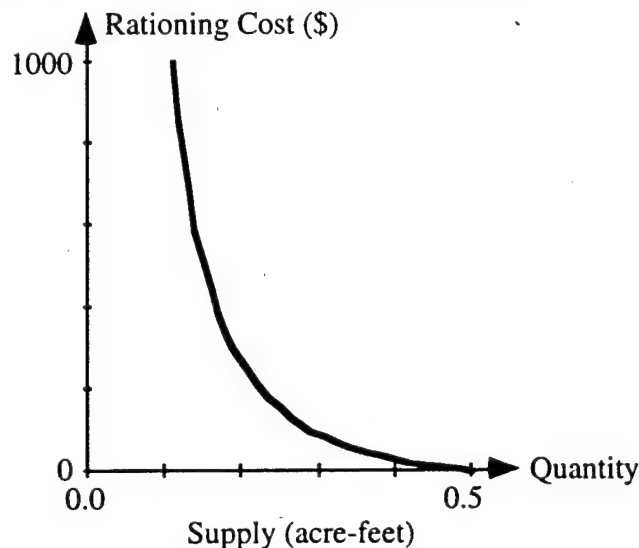
Assuming the system has a demand elasticity of $\alpha = -0.33$, a normal price of \$200 per acre foot, and a normal demand of $Q_0 = 600,000$ acre-feet, then 50% rationing results

in an annual cost (or lost benefit) of \$180,000,000, and 90% rationing results in an annual cost of almost six billion dollars. Considering the severity of rationing, these costs may be reasonable, though it may be easier to evaluate if we consider the impact per family.

We can distribute these costs to estimate the impact per family. A typical American family consumes around $Q_0 = 0.5$ acre-foot annually. Figure 10D1 displays the rationing cost using $\alpha = -0.33$ and a $P_0 = \$200$ per acre-foot. We can see that 50% rationing results in an annual cost of \$150, and 90% rationing results in an annual cost of \$4950. In spite of the high prices, these rationing-cost estimates seem comparable to the damage we might expect from such extreme levels of rationing.

In summary, to estimate the cost of water rationing one should consider the types of consumption and the time over which adjustments can be made. Urban consumption of water appears to be particularly inelastic with changes in price, resulting in high rationing costs. When urban areas have sufficient time to adjust to water conditions, an elasticity of $\alpha = -0.5$ may be appropriate. In the short-term $\alpha = -0.33$ appears reasonable, though there is evidence that water demand can be even less elastic under some conditions.

Figure 10D1. Estimate of Rationing Cost for a Typical Family



CHAPTER 11

OPTIMAL CONJUNCTIVE-USE OPERATIONS AND PLANS

Heuristic or intuitive rules based on experience may not be efficient when applied to the management of water-supply systems that contain both surface and subsurface storage. In particular, rules that assign subsurface storage the role of a back-up supply to surface storage do not recognize the different capabilities of surface and subsurface storage. This is the case if we recharge groundwater only after filling surface reservoirs or if we pump groundwater only after exhausting surface supplies. We demonstrate how to incorporate the different capabilities of surface and subsurface storage in appropriately cautious real-time control of conjunctive-use systems and how to evaluate the benefit of adding groundwater supplies to an existing surface-water supply system. To illustrate this, we use the East Bay Municipal Utilities District of Oakland, California, as an example.

A. INTRODUCTION

Increasing demands on limited water resources are leading managers to consider innovative designs and control methods that can improve system reliability. Conjunctive use—the coordinated management of groundwater and surface water—is an affordable and environmentally sound method for enhancing the reliability of water supply systems [Fisher *et al.*, 1995; Lettenmaier and Burges, 1979]. However, the potential for using the subsurface as a natural storage facility has not been fully recognized, and most large water supply systems continue to depend exclusively on surface water supplies. Managers of many systems view groundwater as providing only a back-up supply used only in times of shortage [Lettenmaier and Burges, 1979].

A barrier to conjunctive-use methods is that it is not clear how best to operate conjunctive-use systems. Surface and subsurface storage have operating costs and constraints that are fundamentally different. On one hand, surface storage can be filled and drained rapidly, while rates of recharge and pumping of groundwater are limited. On the other hand, aquifer storage capacity considerably exceeds available surface storage in

many watersheds [Buras, 1963]. In addition, pumping and recharge may be costly, while surface releases may generate hydropower benefits.

Such differences suggest that control of reservoirs and aquifers should be quite different. For example, the potentially large subsurface storage capacity and the expense of pumping and recharge suggest that we store water in the subsurface as insurance against long-term water needs. In contrast, limited surface storage capacity and the low cost of active management suggest that we use surface water supplies to meet short-term needs during the next season or year. As *Lettenmaier and Burges* observe, "In contrast to the rather long-term failure modes encountered in excessive reliance on groundwater supplies, shorter scale (e.g., annual or seasonal) failures may result from exclusive use of surface supplies. The difference in time scales results because typical surface storage reservoir volumes are much smaller compared to abstractions than are groundwater supplies" [1979, p. 1].

The active use of aquifers for storage within a comprehensive water management program can yield significant benefits. By taking advantage of the distinctly different characteristics of surface and subsurface storage, we may significantly improve conjunctive-use operations and supply reliability. Also, we can identify levels of surface storage and pumping capacity that can be traded off effectively to achieve equivalent levels of reliability [*Lettenmaier and Burges*, 1979, p. 59].

This chapter will identify efficient real-time control policies for conjunctive use using simulation and optimization, or "systems analysis." Systems analysis allows us to identify control policies for new or modified systems for which we do not have sufficient operating experience to rely on heuristic control methods. In particular for conjunctive use systems, systems analysis allows us to identify control policies that efficiently ration water supplies and allocate stored water between surface and subsurface storage.

Active management of groundwater can significantly improve the reliability of water supplies and can reduce expected costs, in spite of pumping and recharge costs. The results of systems analysis demonstrate that the severe impact of water shortages requires us to make active use of both surface and groundwater storage mechanisms. The severity of impacts grows rapidly with the degree of rationing, so we have an incentive to incur modest rationing and operating costs when these reduce the likelihood of more severe future rationing. In other words, efficient real-time control policies include management decisions that "hedge," sacrificing some current water-use benefits to ensure future benefits.

In addition, this chapter will explicitly evaluate the benefit of adding surface and subsurface storage to an existing water supply system. By incorporating capacity

constraints as state variables in system models, we can evaluate the expected benefit of a wide range of expansion alternatives while simultaneously identifying the best operating policies. As a result, we can objectively balance the reliability and operating costs of various options with the capital costs of these options.

B. PROBLEM DESCRIPTION

To illustrate the identification of real-time control policies and capacity expansion benefits, we consider a system model similar to that discussed by *Buras* [1963; 1972] and summarized by *Yakowitz* [1982]. It is based on the system of the East Bay Municipal Utilities District (EBMUD) of California that supplies water to communities along the eastern edge of the central and southern San Francisco Bay [*EBMUB*, 1992; *Fisher et al.*, 1995].

1. The EBMUD System

The EBMUD system serves the residential needs of approximately 1.2 million people, as well as the industrial, commercial, and institutional needs in the East Bay region of the San Francisco Bay area. About 95 percent of its water supply is from the Mokelumne River's 575-square mile watershed on the western slope of the Sierra Nevada Mountains.

Streamflow supplies are seasonal and uncertain, and inflows to EBMUD's reservoir system average 720 thousand acre-feet (TAF) annually (1 TAF = 1.23 million cubic meters). Annual flow has varied between a low of 130 TAF and a high of 1,595 TAF. Average monthly flow varies from a high of over 100 TAF in May to a low of about 30 TAF through the fall months. Typically, a year will have a month of high flow approaching 200 TAF and a month of low flow approaching 10 TAF. At times, the natural flow of the river has ceased.

Future streamflows can be predicted with some accuracy using available information. The intra-annual streamflows on the Mokelumne River are seasonal and autocorrelated, with a monthly correlation coefficient of 0.8 to 0.9. In addition, much of the late spring and summer runoff is a result of melting snow pack, and measurements of snow pack are available throughout the winter. Therefore, consideration of the season, prior streamflows, and measurements of snow pack all contribute to streamflow prediction.

EBMUD manages two reservoirs having a combined capacity of 641 thousand acre feet (TAF) on the Mokelumne River. Up to 200 TAF of this storage is reserved for

flood control (under agreements with the U. S. Army Corps of Engineers), and an additional 21 TAF is dead storage. Besides water supply and flood control functions, these reservoirs also have a combined hydropower generating capacity of 39 megawatts.

An 82 mile aqueduct transports water to the service area for use or for storage in five terminal reservoirs. The aqueduct has a delivery capacity of 200 million gallons per day (mgd) by gravity flow or 325 mgd by pumping. This maximum delivery capacity coincides with EBMUD water-rights permits to divert up to 325 mgd (364 TAF per year) for use in its service district. The terminal reservoirs provide an additional 155 TAF of storage capacity (with 17 TAF of dead storage).

2. Proposed Aquifer Storage

Due to increasing needs for water in the district and in the Mokelumne River basin, EBMUD has been considering a number of options to prevent deterioration of its water supply reliability. These options include adding subsurface storage and increasing surface storage.

Accessible subsurface storage appears substantial. The Mokelumne River and EBMUD Aqueduct run west from the Sierras across the Central Valley of California. This area is underlain by extensive fresh-water bearing formations of thick sand and gravel totaling a few hundred feet in the East and increasing to almost two thousand feet near the Delta. Well capacities are frequently 500 to 1500 gallons per minute (gpm), with specific capacities of 35 to 60 gpm per foot and transmissivities of 60,000 to 80,000 gallons per day per foot [DWR, 1967]. Also, there is a great amount of available space for aquifer storage because significant development of groundwater for local agricultural and municipal needs has depressed the water table by an average 50 to 100 feet below pre-development levels [DWR, 1967; EBMUD, 1992].

3. System Model

The conjunctive-use model that we analyze in this chapter is a simplified representation of the EBMUD system with an added aquifer-storage component (Figure 11B1). This model is not intended for use in systems analysis to solve specific operating or planning problems for EBMUD, but to illustrate some of the general characteristics of optimal control and planning of a conjunctive-use system. The simple conjunctive-use model preserves essential components of a conjunctive-use system without including seasonality or complex system structures. While these may be important for systems analysis applied to the real system, they complicate our ability to observe and evaluate the general characteristics of optimal control and planning. In the next section and the

concluding remarks, we will summarize additional model components appropriate to solve specific operating or planning problems for EBMUD.

Variables used to quantify the conjunctive-use model consist of control variables, \mathbf{u} , state variables, \mathbf{x} , and stochastic variables, \mathbf{s} . Control variables include decisions to supply water to meet demands and to allocate water between surface and subsurface storage. State variables define the amount of water currently stored separately in surface and subsurface reservoirs. Stochastic variables define reservoir inflows. Table 11B1 summarizes the variables of the simple conjunctive-use model and bounds on these variables. Units are in thousands of acre feet.

The state of the system evolves under the influence of control decisions and reservoir inflows according to the linear transition equation

$$\mathbf{x}_{(t+1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_{(t)} + \begin{bmatrix} -1 & 1 & -1 & -1 \\ 0 & -1 & 1 & 0 \end{bmatrix} \mathbf{u}_{(t)} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{s}_{(t)} \quad (11B1)$$

This equation describes the change in surface and subsurface storage levels during any year-long stage t . The ending surface storage is the sum of beginning storage $x_{1,(t)}$, inflow $s_{1,(t)}$, and groundwater pumping $u_{2,(t)}$, minus water supplied to users $u_{1,(t)}$, groundwater recharge $u_{3,(t)}$, and release downstream $u_{4,(t)}$. The ending subsurface storage is the sum of beginning storage $x_{2,(t)}$ and recharge minus pumping. In addition, the state of the system evolves subject to the bounds in Table 11B1 that constrain feasible decisions and attainable states.

Figure 11B1. The Simple Conjunctive-Use System

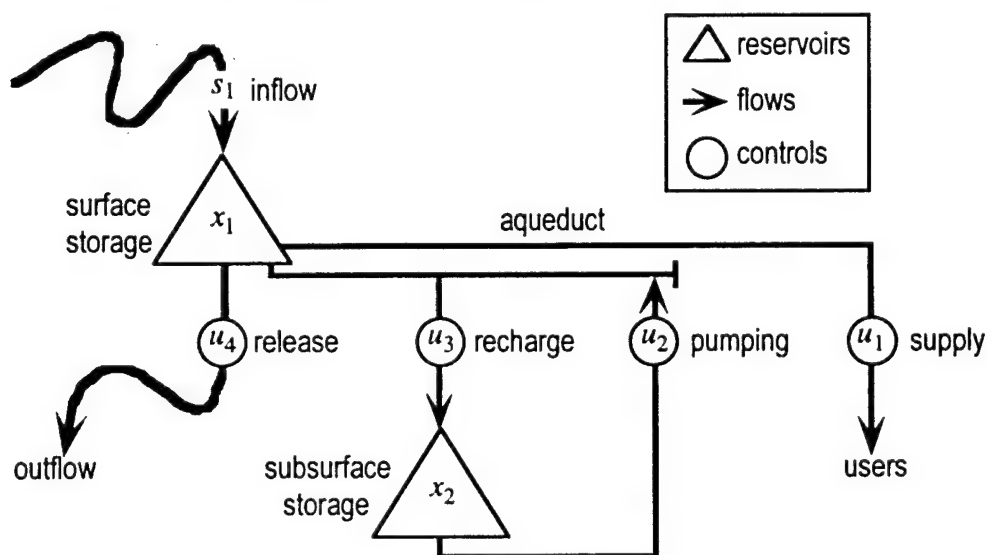


Table 11B1. Variables of the Simple Conjunctive-Use Model

Variable	Type	Definition	Min.*	Max.*
u_1	control	supply to users (annual)	0	600
u_2	control	groundwater pumping (annual)	0	100
u_3	control	groundwater recharge (annual)	0	50
u_4	control	release downstream (annual)	0	infinite
x_1	state	surface reservoir storage	0	200
x_2	state	subsurface reservoir storage	0	500
s_1	stochastic	inflow from streams	0	infinite

* units in thousands of acre feet

4. Demand, Streamflow, and Storage

The effective demand that EBMUD must satisfy in managing its system is greater than the demand in its district summarized earlier. Because EBMUD's water rights are junior to those of most other users, it must manage its system to meet demands of all users to ensure that it can meet demands in its own district. Demands of these senior users are currently about 100 TAF annually, but these may significantly increase because of growth and new streamflow maintenance requirements [EBMUD, 1992]. Annual demands for Mokelumne water supplies are expected to increase to perhaps 600 TAF in twenty or thirty years. Thus, annual demands in the model are a constant 600 TAF.

Streamflows in the model are lognormally distributed with a mean annual flow of 700 TAF annually and a standard deviation of 350 TAF. For simplification, autocorrelation and snow-pack measurements are neglected as they have a minor impact on predicting streamflows a year in advance.

The simple conjunctive-use model uses year-long stages, so we require annual updates of supply and allocation decisions. Though the actual real-time control of the EBMUD system requires more frequent updates of control decisions, our use of year-long stages allows us to illustrate more clearly the impact that differences in surface and subsurface storage have on real-time control and on evaluation of capacity expansion options. Because existing EBMUD system storage is used to regulate both annual and seasonal variability of Mokelumne River flows, an annual model greatly reduces the impact that streamflow variability has on water supply reliability. To ensure that water shortages have a reasonable impact on system operations, the model uses a reduced surface storage capacity of 200 TAF. In contrast, subsurface storage need not be reduced

since limited pumping and recharge capacities (and associated costs) means that subsurface storage has a smaller impact on seasonal variability.

Subsurface storage capacity is 500 TAF. This is significantly greater than the surface storage capacity of 200 TAF but still less than the potential storage capacity of aquifers in the Central Valley. It is likely that EBMUD's access to subsurface storage would be constrained by legal and contractual arrangements and not by the physical capacity of the aquifers, so a simple bathtub model for groundwater may be sufficient. However, a more accurate representation of subsurface flow may be required under other conditions and for other systems. For example, EBMUD might use the smaller subsurface storage capacity in its own service district where it has greater access and control. In this case, local dewatering of the aquifers can make it impossible to fully access the available storage capacity.

To realistically solve an actual operating or planning problem for the EBMUD system, it may be necessary to use month-long or week-long stages and to model each reservoir separately using actual capacities. However, shorter stages increase the total number of stages required to span the operating horizon of the problem, and added reservoirs and stochastic inputs increase the number of state variables. All these increase the computational effort required to solve a problem: effort grows linearly with the number of stages and exponentially with the number of state variables. Although, we can solve these more-complex problems using the systems analysis methods developed in this thesis, we will concentrate on the simpler model.

5. Value Model

In each stage, the cost of control decisions, u , is a sum of costs

$$C(u) = \text{shortage cost} + \text{pumping cost} + \text{recharge cost} \quad (11B2)$$

Shortage cost is the loss from rationing that leaves unsatisfied some demands for water and is a function of the water supply decision u_1 . Pumping and recharge costs are operating costs that result from allocating stored water between surface and subsurface storage by a decision to pump u_2 or recharge u_3 .

SHORTAGE COST

Identification of shortage costs can be difficult. As a result, few reservoir management studies attempt to estimate actual shortage costs in spite of the important impact that these costs have on hedging. For example, *Fisher et al.* [1995] compare the impact of various capacity expansion alternatives of the EBMUD system by evaluating

only the capital and operating costs of the various alternatives. However, the impact of shortages on operations and plans may not be correctly anticipated without a reasonable estimate of the shortage costs.

In contrast, this study makes explicit use of a shortage-cost estimate to identify the benefits of water use and the cost of rationing. These costs are evaluated by a cost function that assumes that the “willingness to pay” for the use of water is a reasonable surrogate for the benefits of water supply [Dandy, 1992]. In addition, the cost function assumes that the elasticity of demand (i.e., its sensitivity to changes in price) for water is constant over a wide range of prices.

We can identify a cost function given these assumptions and some additional data on the system. Using the results of Chapter Ten, we assume an elasticity of $\alpha = -0.33$ and a price of \$200 per acre-foot when we supply the full annual demand of $u_1^{\max} = 600$ TAF. These values are reasonable given elasticities estimated by other authors [Billings and Agthe, 1980; Danielson, 1979; Martin and Thomas, 1986; Mercer and Morgan, 1989; Moncur, 1987; Moncur, 1989] and recent California water prices [CUWA, 1991; EBMUD, 1992; Fisher et al., 1995; Howitt, 1994; McClurg, 1992a; McClurg, 1992b]. Using these data in the general cost-function equation [Dandy, 1992], the total shortage cost in a year-long stage is

$$\text{shortage cost} = \$60,000,000 [(u_1/u_1^{\max})^{-2} - 1] \quad (11B4)$$

Under “normal” conditions, there is no rationing and $u_1/u_1^{\max} = 1$. Under water-shortage conditions, rationing may be required due to either hedging or unavoidable shortages and $u_1/u_1^{\max} < 1$.

PUMPING COST

To provide an annual pumping capacity of 100 TAF, EBMUD could develop a field of perhaps 50 to 100 large capacity wells with total capacity of about 100,000 gpm. For pumping lifts of approximately 100 feet around the Mokelumne River and the aqueduct, the cost of electricity [Georgakakos and Vlatsa, 1991] is about \$10 to \$20 per acre-foot. These costs could increase as much as 50% when pumping at a maximum rate based on specific well capacities. In addition, we can expect additional costs for operation and maintenance for the well field.

For this study, we assume a marginal cost for pumping of \$40 per acre-foot that increases linearly to \$80 per acre-foot when pumping at the maximum annual rate of $u_2^{\max} = 100$ TAF. At this maximum pumping rate, the average cost of pumping is \$60 per acre-foot.

Using these costs, the total pumping cost in a year-long stage is

$$\text{pumping cost} = \$4,000,000 (u_2/u_2^{\max}) [1 + 0.5 (u_2/u_2^{\max})] \quad (11B5)$$

This cost should be sufficient to discourage unreasonable active pumping of groundwater and should allow us to identify water supply conditions that realistically justify pumping decisions. This does not include capital costs for installation and development of the well field. For example, *Fisher et al.* [1995] estimate costs of drilling and pump installation to be about \$25,000 per well.

RECHARGE COST

The marginal cost of recharge is highly variable, depending on the methods used and the character of sites available for implementing a recharge program. Methods may include surface spreading, injection, and enhanced natural recharge (e.g., by structural and institutional arrangements that replace groundwater pumping with surface supplies during wet years). For example, the amount of land required and the maintenance of this land for recharge is highly variable, depending on soil type, infiltration rate, and subsurface geology.

For this study, we assume a constant marginal cost for recharge of \$40 per acre-foot. Depending on the recharge method used, this cost could represent the purchase, transportation, or treatment of water prior to recharge, or might represent the cost of recharge-facility maintenance. The total recharge cost in a year-long stage is

$$\text{recharge cost} = \$2,000,000 (u_3/u_3^{\max}) \quad (11B6)$$

As with pumping cost, this cost should be sufficient to discourage unreasonable active recharge of groundwater and should allow us to identify water supply conditions that realistically justify recharge decisions. This does not include costs for land or other capital costs for installation of recharge facilities. These capital costs are also highly variable because of a diversity of recharge options and the cost of these options. For example, the cost of land is highly variable. Also, the cost of establishing and maintaining other structural and institutional arrangements can be significant due to government regulations and the diversity of interests that must be considered.

C. EVALUATION OF OPERATIONS AND PLANS

By using mathematical models that simulate the structure and dynamics of reservoir systems, we may simulate and evaluate proposed management options

conveniently. Also, by applying optimization methods, we may quickly and efficiently identify the best options for control and planning when a range of options exists. For the simple model, the best options are those that achieve the lowest expected cost of rationing and operations modeled above.

The real-time control of a conjunctive-use system includes operating decisions that ration water supplies to users and that allocate stored water between reservoirs. For these controls to be efficient, we need to include information about the system that identifies its state. For the simple model presented above, this information consists of current surface and subsurface storage levels and the forecast of the current year's inflows.

The planning of a conjunctive-use system includes analysis and implementation of changes in system configuration, inputs, or goals. To evaluate accurately the expected costs and benefits of these changes, we need to identify new control policies to operate the system. For the simple model presented above, we will identify control policies and expected costs for a range of pumping, recharge, and surface-storage capacities.

As we observed in Chapter Nine, we desire control policies that consider the impact that current decisions have on future costs. This is important because current control decisions limit future management options. As a result, managers have an incentive to sacrifice some of the current performance (i.e., by incurring short-term costs) to improve future performance. By trading some short-term benefits for long-term benefits, managers hedge. For example, during a drought, a system manager may ration water supplies to reduce potential damages from more severe shortages in the future.

Hedging is important when variable and uncertain inputs drive the state of a dynamic system and the marginal cost (per unit of deviation) from the target is a non-linear function of the deviation. For example, conjunctive-use systems are driven by inputs of water supply and demand that often are highly variable and uncertain, and the marginal cost of shortages can increase dramatically and non-linearly as a function of rationing severity. When inputs are uncertain, appropriate control decisions cannot be identified by a pre-determined control schedule because some information about future streamflows and other stochastic inputs is not available. Instead, we make "real time" decisions that use information that is available when the decisions are made. Thus, the problem is how to optimize control policies rather than control schedules.

Our goal is to identify control policies that minimize the combined expected costs of water rationing and system operations. Total costs accumulated over a multi-year time horizon are

$$V = \sum_{i=1}^N C_{t_i} (1-r)^i \quad (11C1)$$

for an N year horizon using a discount rate r .

To solve the control policies of the conjunctive use problem, we apply discrete dynamic programming using the second-order GDP method (Chapter Five). In each stage, we calculate expected costs as a probability weighted sum of outcomes using Gaussian quadrature (Chapter Seven). We use foresight of the current year's inflows to identify control decisions as this more closely represents real operations that are based on a sophisticated prediction of streamflow. Also, real operations are updated more frequently than permitted in the simple model.

The following results are for the first stage of a hundred-year time horizon using a 4% annual discount rate ($r = 0.04$). In effect, these results identify the infinite-horizon, steady-state solution that does not depend on boundary conditions at the end of the time horizon. Because system conditions (e.g., system constraints and hydrology) are static, we observe that control policies and the cost function converge to steady state values after a few decades. This convergence is promoted by the discount rate; however, as we will see later, the impact of a discount rate has only marginal impacts on control policies and capacity expansion benefits.

D. RESULTS FOR REAL-TIME OPERATIONS

The solution of the simple conjunctive-use problem provides a set of control policies for system operation. The solution also provides the expected total cost of water rationing, pumping, and recharge that results from application of these policies. Control policies are expressed as functions of the initial state (i.e., initial surface and subsurface storage levels) and the current year's inflows, and cost is expressed as a function of the initial state.

Because we use foresight of current inflows to identify control decisions, the actual division between initial surface storage $x_{1,(t_1)}$ and current inflow $s_{1,(t_1)}$ does not influence control decisions and the expected cost. We can define "available surface water" as the sum $(x_1 + s_1)_{(t_1)}$ to simplify illustration of the control-policy and cost-function solutions. Note that with this definition, available surface water has no maximum value and we will only illustrate results for available surface water less than 1000 TAF. We also define "available groundwater" simply as subsurface storage $x_{2,(t_1)}$.

1. Supply Policy

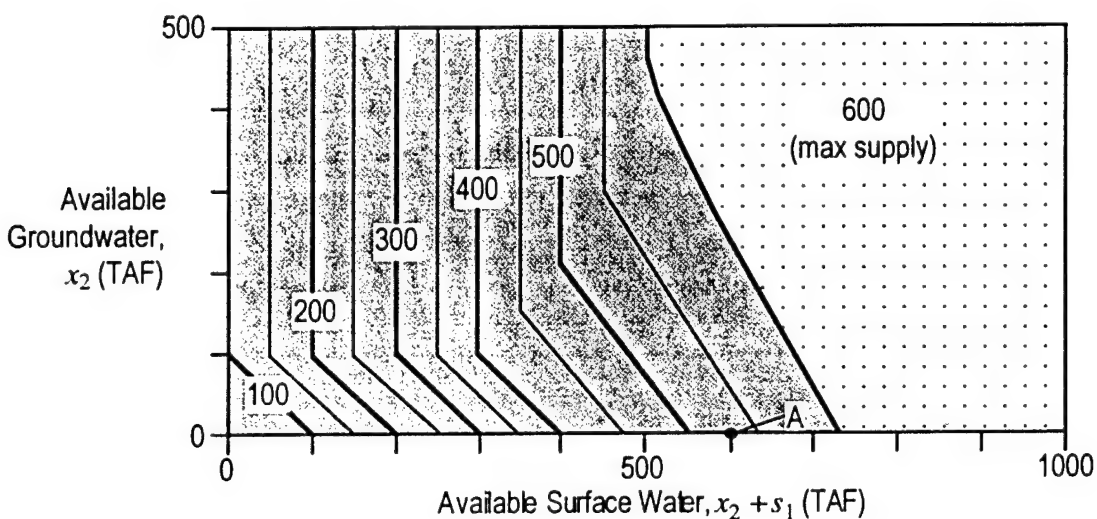
Figure 11D1 depicts the supply policy (i.e., release to users $U_{1,(t_1)}$) as a function of surface water and groundwater levels. Supply is in thousands of acre feet per year and levels are in thousands of acre feet. As surface-water and groundwater levels increase, we see that the supply increases until a plateau at the maximum of 600 TAF per year.

Because annual pumping is limited to 100 TAF, rationing is required when surface water supply is low even if sufficient groundwater supply is available. The release to users must be less than the sum of available surface water and pumping. In Figure 11D1, this is indicated by vertical portions of the contours. When available groundwater is below 100 TAF, there is insufficient groundwater for maximum pumping.

Figure 11D1 also indicates that we should ration even when sufficient surface water is available. At point A for example, there is sufficient water to meet all demands with 600 TAF available surface water. Instead, only about 530 TAF is released to users and 70 TAF is stored to hedge against future shortages. Available water can be released to meet current demands or can be stored to meet future demands, and we should balance these two uses under certain conditions.

In Figure 11D1, we can see that hedging exists when surface-water level is moderately low (approximately 400 to 730 TAF) and groundwater level is low. There is an incentive to ration under these conditions, even if supplies and pumping capacity are sufficient to meet all current demands. These decisions are examples of how we employ cautious management to balance the cost of current rationing with the potential cost of future, more severe rationing.

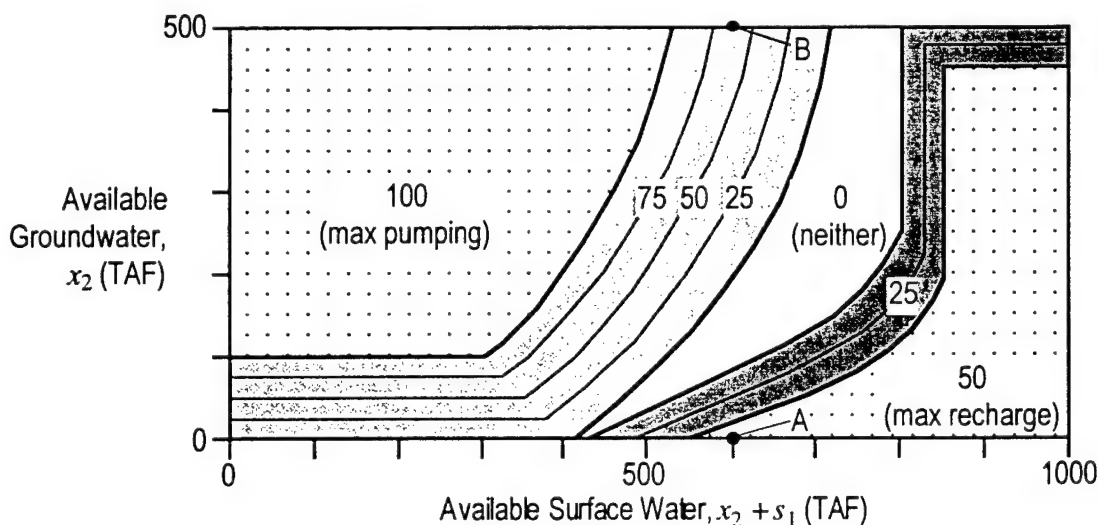
Figure 11D1. Supply Policy: Release (TAF per Year) to Users as a Function of Available Water



2. Allocation Between Surface and Subsurface Storage

Figure 11D2 depicts the pumping and recharge policies ($U_{2,(t)}$ and $U_{3,(t)}$) that transfer water from and to groundwater as a function of surface water and groundwater levels. In effect, these policies allocate stored water between surface and subsurface storage. As in Figure 11D1, pumping and recharge are in thousands of acre feet per year and levels are in thousands of acre feet. As the surface-water level decreases and the groundwater level increases, we see that pumping increases until a plateau at the maximum 100 TAF per year. In contrast, as the surface-water level increases and the groundwater level decreases, we see that recharge increases until a plateau at the maximum 50 TAF per year.

Figure 11D2. Pumping and Recharge Policies: Transfers (TAF per Year) from and to Groundwater as a Function of Available Water



PUMPING POLICY

Maximum pumping is required whenever available surface water is low and groundwater is available. This allows us to provide the maximum supply possible during extreme water-shortage conditions. In Figure 11D2, this is indicated by horizontal portions of the pumping contours. When the subsurface storage is below 100 TAF, pumping cannot exceed available groundwater.

It might be thought that pumping should be determined solely by the need to meet current demands. However, Figure 11D2 indicates that we should not necessarily pump at the maximum rate needed to meet current demands when groundwater is low. Instead, we should reduce pumping when available surface water increases beyond 300 to 400

TAF and the groundwater level is low. These decisions coincide with supply decisions that hedge (Figure 11D1). This means that under conditions of moderate rationing to save water for future use, we should reduce or cease pumping to store some of this water in the subsurface. The appropriate level of pumping depends on the available groundwater in storage.

Likewise, Figure 11D2 indicates that we should not necessarily cease pumping if not needed to meet current demands when the available groundwater level is high. Instead, we should continue pumping when available surface water increases beyond 600 TAF and the groundwater level is high. For example, when there is 600 TAF available surface water and 500 TAF groundwater (point B in Figure 11D2), there is no rationing (see Figure 11D1) and there is sufficient available surface water to meet all demands. Instead of halting pumping under these conditions, pumping is continued at a moderate rate to transfer a greater portion of the remaining stored water to surface storage where it is more readily available. This means that it is desirable to shift some water from subsurface storage to surface storage when the groundwater level is high and surface-water level is low. Because the pumping rate is constrained, there is a risk of severe rationing if future inflows are extremely low and surface reservoirs are empty.

RECHARGE POLICY

Maximum recharge is required whenever available surface water is high and groundwater is low. This is especially true when the surface-water level exceeds 800 TAF (the sum of maximum demand and surface storage); excess water above this level must be released “unused” downstream unless it is recharged. In Figure 11D2, this is indicated by the vertical portions of the recharge contours. When subsurface storage is above 450 TAF, recharge cannot exceed available subsurface storage.

It might be thought that recharge should be determined solely by the availability of excess water. However, Figure 11D2 indicates that we should not necessarily recharge only after meeting current demands when groundwater is low. Instead, we should recharge when available surface water is below 800 TAF and the groundwater level is low. In particular, we should also recharge when available surface water is low enough that supply decisions hedge (Figure 11D1). This means that under conditions of moderate rationing to save water for future use, we should recharge to store some of this water in the subsurface. As with pumping, the appropriate level of recharge depends on the available groundwater in storage. For example, when there is 600 TAF available surface water and zero groundwater, we should ration to store 70 TAF for future use (point A, Figure 11D1). However, we also should recharge at the maximum rate (point

A, Figure 11D2). This means that we have a strong preference for storing water in the subsurface when groundwater level is low. If the surface reservoir fills in the following year, then we will lose the future benefit from current efforts to save water by rationing.

ALLOCATION

In effect, pumping and recharge decisions allocate stored water between surface and subsurface storage. In the absence of operating costs, pumping and recharge policies should seek allocations that give the best water supply reliability. These allocations should be based on the different capabilities and limitations of the storage mechanisms and the expected benefit of stored water: water stored in the surface may be "lost" if the reservoir subsequently fills; water stored in the subsurface may be inaccessible during severe shortages.

In this simple conjunctive-use system, we never have an incentive to pump and recharge simultaneously. Indeed, water-supply conditions under which we should pump and recharge may be separated by a significant gap. In Figure 11D2, this is indicated by the unshaded gap between the pumping and recharge contours.

Because of operating costs, we should pump and recharge to achieve a better allocation only when the benefit of an improved allocation exceeds the cost of achieving that allocation. The benefit of improved allocation is large when water levels are low, and we see that the gap between pumping and recharge is small. As water levels increase, the benefit of improved allocation decreases and the gap widens.

When supplies are sufficiently large though, the gap again narrows as we recharge to store water that would otherwise be released downstream. The benefit of recharging this water is much greater than the benefit of recharging water that otherwise could be stored in the surface reservoir.

3. Downstream-Release Policy

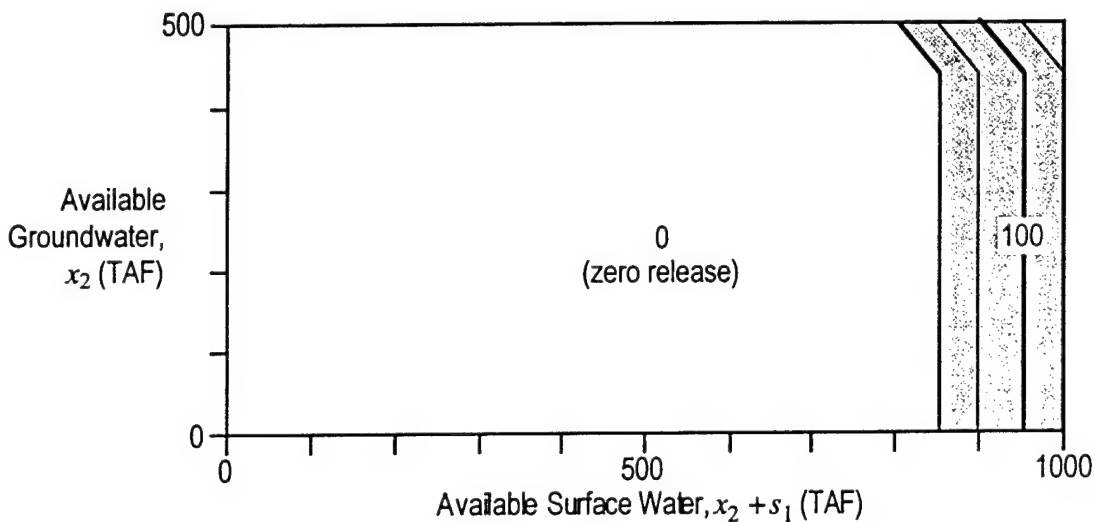
Figure 11D3 depicts the downstream release policy $U_{4,(t)}$. As before, release is in thousands of acre feet per year and levels are in thousands of acre feet. Water is released downstream only when available surface water levels are high, and the bend in these contours above 450 TAF is because there is insufficient subsurface storage capacity for maximum recharge.

The policy for downstream release is somewhat obvious since the control policy always saves as much water in surface and subsurface storage as possible. In the simple conjunctive-use model, rationing is frequently required because demands are such a large fraction of average streamflow. As a result, the expected cost of rationing always

outweighs the cost of recharge, and a downstream release occurs only when water levels exceed demands and opportunities for storage.

Because the system objective includes only the goals of minimizing water rationing and operating costs, there is no benefit from releasing water downstream except when required by capacity constraints. However, if we were to add other goals such as flood control, release decisions would also show significant hedging to maintain available empty storage as a buffer against peak inflows.

Figure 11D3. Release Policy: Release (TAF per Year) Downstream as a Function of Available Water



4. Expected Cost of System Operations

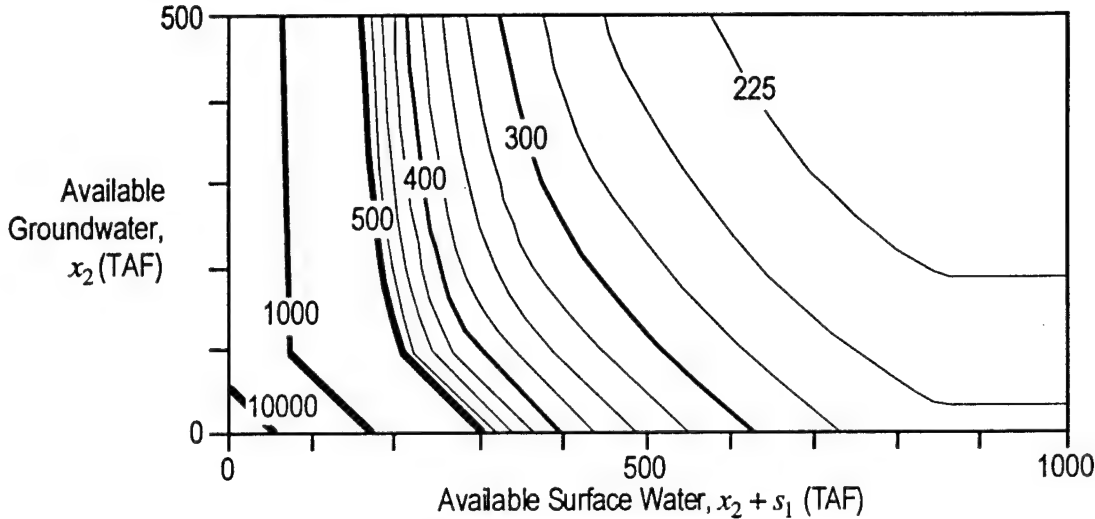
Figure 11D4 depicts the total discounted cost as a function of available surface water and groundwater levels. Cost is in millions of dollars and levels are in thousands of acre feet. As surface-water and groundwater levels decrease, we see that the total cost increases and becomes infinite as levels approach zero. Zero supply means that we have no water for any use, even for sustaining life.

In reality, we should not expect to see the high costs that result from supplies approaching zero. It is extremely unlikely that we will operate with critically low supplies because (1) extremely low inflows are unlikely, (2) storage levels should rarely approach zero with cautious management, and (3) alternate supplies are often available when prices are sufficiently high (i.e., hauling water by trucks, desalination, and other supply methods become cost effective).

Figure 11D4 shows that expected costs do not change when available surface water is greater than 850 TAF because we can neither use nor store supplies greater than

this amount. Such large surface supplies permit us to fully meet maximum demands of 600 TAF while also permitting us to fill the surface reservoir (200 TAF) and recharge at the maximum annual rate (50 TAF).

Figure 11D4. Expected Total Cost (Million \$) as a Function of Available Water Using Foresight of Current Year's Inflows



5. Cost-To-Go

Figure 11D4 plots the expected total cost of rationing and operations as a function of surface water and groundwater levels assuming that we know current inflow $s_{(t_1)}$. We evaluate the cost as an expectation over all subsequent inflows $\{s_{(t_2)}, \dots, s_{(t_N)}\}$. In contrast, Figure 11D5 depicts the expected total cost F as a function of initial surface and subsurface storage levels x_1 and x_2 assuming no foresight. We evaluate this cost as an expectation over all current and future inflows $\{s_{(t_1)}, \dots, s_{(t_N)}\}$. This function is known as the "cost-to-go" because it describes the expected cost to go from any initial state of the system. The cost-to-go function is useful in dynamic programming because it describes expected cost strictly as a function of the state. Thus, we can use the cost-to-go function to evaluate the impact that current control decisions have on future costs through the effect they have on the state of a system.

We can evaluate an expected cost in Figure 11D4 that assumes foresight by using the cost-to-go function $F_{t_2}(\mathbf{x})$ (which is the same as $F_{t_1}(\mathbf{x})$ because of solution convergence using a discount rate). Using foresight of current stochastic inputs \mathbf{s} ,

$$f_{t_2}(\mathbf{x}, \mathbf{s}) = \min_{\mathbf{u}_{(t_1)}} \{ C_{t_1}(\mathbf{u}, \mathbf{s}) + F_{t_2}(\mathbf{x}) \} \quad (11D1)$$

where $f_{t_1}(\mathbf{x}, \mathbf{s})$ is the cost function of Figure 11D4. In a recursive fashion, $F_{t_1}(\mathbf{x})$ can be evaluated from $f_{t_1}(\mathbf{x}, \mathbf{s})$ as the probability weighted sum of current inflows,

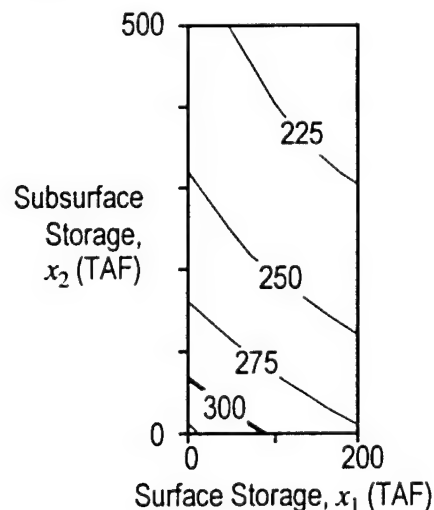
$$F_{t_1}(x) = \sum_s W(s) f_{t_1}(x, s) \quad (11D2)$$

where $W(s)$ is the weight applied to input s . Compared to Figure 11D4, the costs of Figure 11D5 are less variable because of the “averaging” over possible values of current inflow $s_{(t_1)}$. We can see that the results of Figure 11D5 are consistent with Figure 11D4. For example, the overall cost of about \$250 million in Figure 11D5 is consistent with the costs in Figure 11D4 when available surface water is about 700 TAF. Surface water of about 700 TAF represent an “average” condition of streamflow and storage.

Figure 11D5 shows that, using a discount rate of 4%, the total cost of future operations is roughly \$250 million. The annual cost is 4% of the total cost, or roughly \$10 million. These costs decrease as levels increase, but costs are still significant even when reservoirs are initially full. The time horizon is sufficiently long and the discount rate sufficiently low that we cannot avoid or discount the cost of future shortages.

Figure 11D5 also shows that the decrease in costs with increasing levels is not constant. The decrease in costs is greatest when storage levels are low. Furthermore, the decrease in costs depends on the initial surface and subsurface storage levels. As we saw in the supply policy results, the change in cost with storage is significant enough to encourage a balancing of current benefits of water supply (to avoid current rationing) with expected future benefits of storing water (to reduce the impact of future shortages). As we saw in the allocation results, the change in cost with allocation between surface and subsurface storage is significant enough to encourage operations that move toward the best allocations.

Figure 11D5. “Cost-To-Go”: Expected Total Cost (Million \$) from Future Inflows as a Function of Initial Storage Levels



E. RESULTS FOR CAPACITY EXPANSION

We can extend the simple conjunctive-use model to evaluate the effect of changing system capacities. This is accomplished by converting the bounds on surface storage, pumping, and recharge from fixed values to variables. To allow these variable capacities, we add three additional state variables to the simple conjunctive-use model. These three state variables represent added pumping capacity x_3 , added recharge capacity x_4 , and increased surface storage capacity x_5 . We add these state variables to the transition function of equation (11B1), setting each identically equal to its value in the previous stage. Also, we ensure control policy solutions are valid for these capacities by adding constraints to bound pumping and recharge decisions and to bound the surface storage level (Table 11E1).

Changing system capacities in this manner allows us to compare the benefits and trade-off of increasing surface storage and/or adding subsurface storage. The advantage of this approach is that we can evaluate the benefits of an infinite number of capacity expansion alternatives while simultaneously identifying the appropriate real-time control policies that should be applied. As a result, we need not rely on a trial-and-error approach to identify the best capacity expansion alternatives, each requiring a separate control-policy and expected-cost solution.

It is difficult to view the entire solution of this planning problem because the cost function and control policy are functions of five state variables and of current inflows. Instead, we have identified a few representative comparisons of expansion alternatives to view the benefits and trade-off of increased capacities. We are interested in comparing the benefits of capacity expansion alternatives described by state variables x_3 , x_4 , and x_5 , so we evaluate these alternatives assuming surface storage and subsurface storage are nearly full initially ($x_{1,(t_1)} = 200$ TAF and $x_{2,(t_1)} = 500$ TAF). As we will see however, the application of a discount rate means that initial conditions influence the expected benefits, so we also evaluate the alternatives assuming reservoirs are empty initially ($x_{1,(t_1)} = x_{2,(t_1)} = 0$). In addition, we evaluate the cost as an expectation over all current and future inflows using the cost-to-go function, so costs are not a function of current inflows.

Table 11E1. Variables for Capacity Expansion of the Simple Conjunctive-Use Model

Variable	Type	Definition	Min.	Max.
u_1	control	supply to users (annual)	0	600
u_2	control	groundwater pumping (annual)	0	x_3
u_3	control	groundwater recharge (annual)	0	x_4
u_4	control	release downstream (annual)	0	infinite
x_1	state	surface reservoir storage	0	x_5
x_2	state	subsurface reservoir storage	0	500
x_3	state	pumping capacity (annual)	0	100
x_4	state	recharge capacity (annual)	0	50
x_5	state	surface-reservoir capacity	200	300
s_1	stochastic	inflow from streams	0	infinite

* units in thousands of acre feet

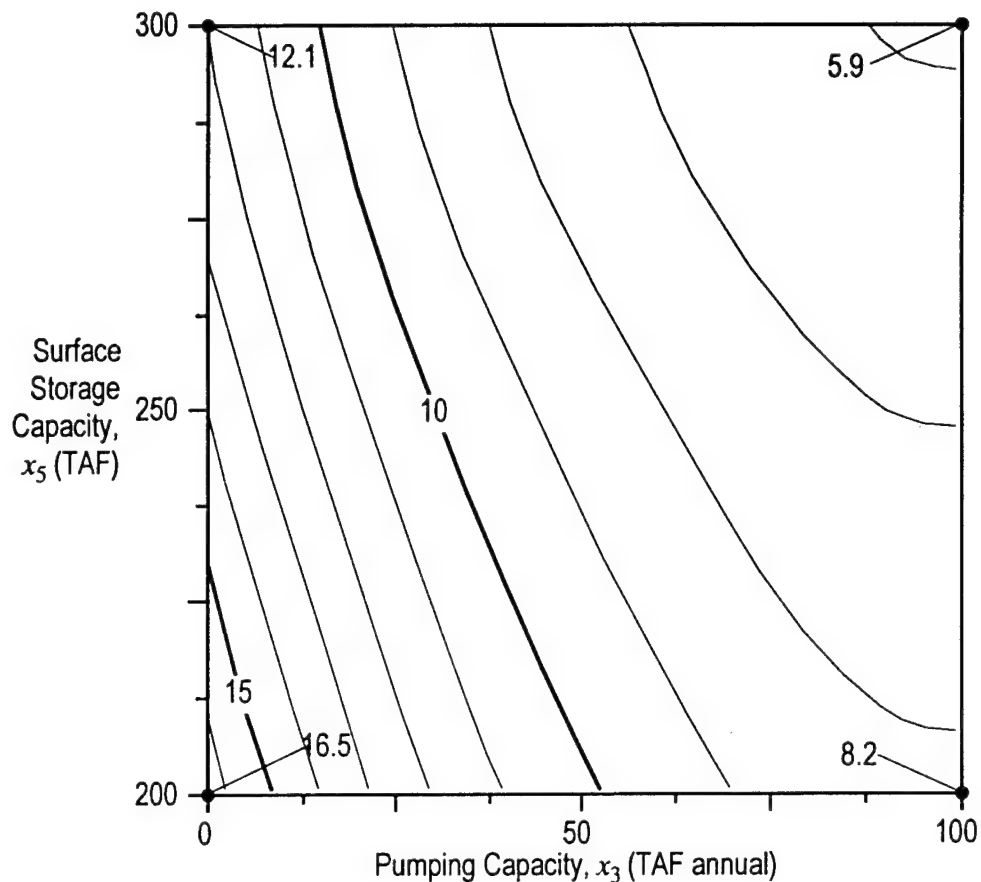
1. Benefits of Groundwater Development

Figure 11E1 depicts the expected annual cost of rationing and system operations as a function of pumping and recharge capacities. Cost is in millions of dollars per year and capacities are in thousands of acre feet per year. As capacities increase, we see that the annual expected cost decreases from \$16.5 million to \$8.2 million. Using the 4% discount rate, the total expected cost for all future operations decreases from \$412 million to \$205 million (annual cost divided by 0.04). These results are for a surface storage capacity of 200 TAF.

Besides identifying the benefit of groundwater development, we can use these results to identify the best level of development and the optimum trade-off between pumping and recharge capacities. The best mix of capacities depends on a balance between the benefits and costs of building and operating pumping and recharge facilities. For the conjunctive-use system, we should balance the benefit of lower rationing costs with the cost of system operations and capital.

We can use the results of Figure 11E1 to identify how to balance these benefits and costs. Figure 11E1 identifies the rationing and operating costs as a function of pumping and recharge capacities, but does not include the capital costs (e.g., cost of installing pumps, buying land, permitting, etc.). In real applications, capital costs are highly variable and site specific, so we simplify the illustration by assuming that marginal costs for expanding pumping and recharge facilities are equal and constant. In other words, each unit of additional capacity costs the same for both. Annualized capital costs

Figure 11E2. Expected Annual Cost (Million \$) for Different Levels of Conjunctive Development with Initially Full Reservoirs



3. Impact of Initial Conditions on Results

Because we apply a discount rate to the cost of future rationing and system operations, the impact of current costs is greater than the impact of future costs. As a result, expected costs are lower if initial conditions are favorable than if initial conditions are unfavorable. In our simple conjunctive use system, conditions are favorable if storage levels are high and are unfavorable if storage levels are low.

Figure 11E3 depicts the expected annual cost as a function of pumping and recharge capacities assuming that reservoirs are empty initially ($x_{1,(t_1)} = x_{2,(t_1)} = 0$). As capacities increase, we see that the expected annual cost decreases from \$19.0 million to \$13.3 million. These costs are higher than in Figure 11E1 where we assume that reservoirs are nearly full initially. Also, the annual costs for full development of groundwater—\$8.2 million in Figure 11E1 and \$13.3 million in Figure 11E3—bracket the average annual cost-to-go in Figure 11D5 of \$10 million.

If we again assume that marginal costs for expanding these capacities are equal and constant as in Figure 11E1, the shaded line in Figure 11E3 identifies the optimum trade-off of pumping and recharge capacities. In this case, we see that the best mix includes more recharge capacity, especially at lower levels of development. Without the ability to mine groundwater, pumping capacity is worthless without some ability to recharge. However, at high levels of groundwater development, initial storage levels become less significant and the best mix approaches that of Figure 11E1.

Figure 11E4 depicts the expected annual cost as a function of pumping and surface storage capacities assuming that reservoirs are empty initially. As capacities increase, we see that the expected annual cost decreases from \$19.0 million to \$10.9 million. Costs are higher than in Figure 11E2. Also, the benefit of added pumping capacity is less because we first must recharge so that we have water to pump.

Figure 11E3. Expected Annual Cost (Million \$) for Different Levels of Groundwater Development with Initially Empty Reservoirs

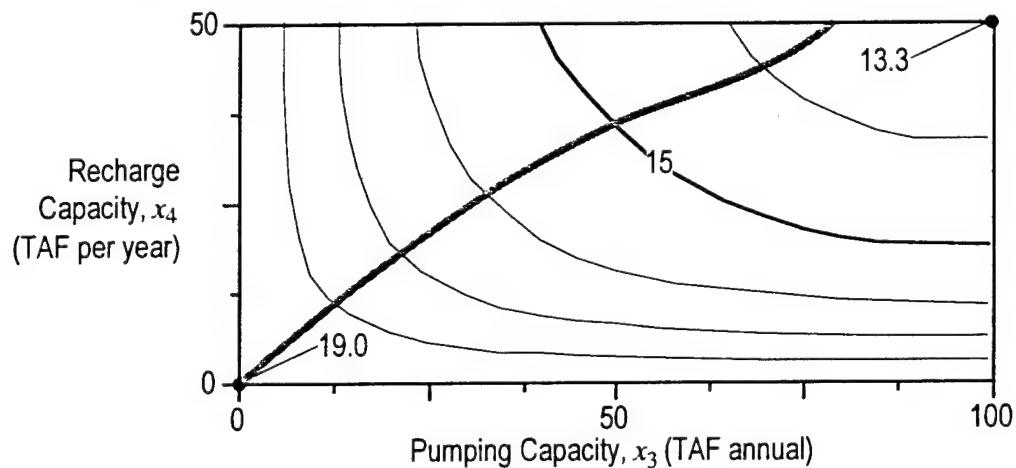
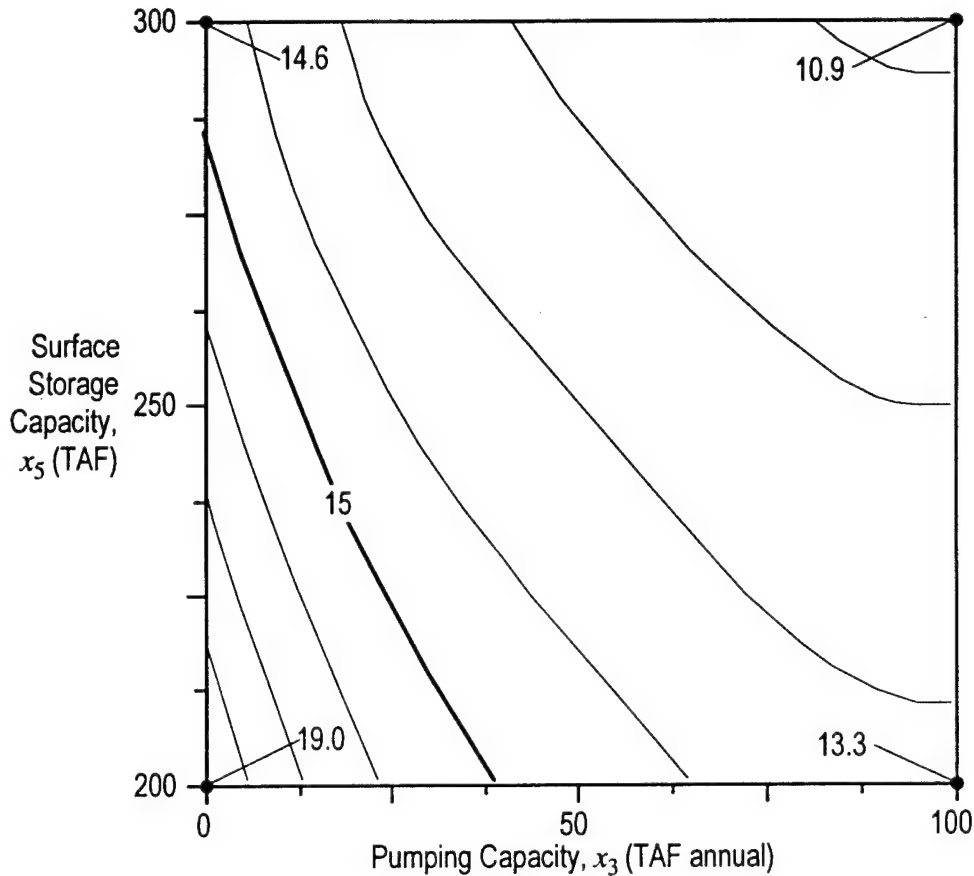


Figure 11E4. Expected Annual Cost (Million \$) for Different Levels of Conjunctive Development with Initially Empty Reservoirs



4. Impact of Discount Rate on Results

For the results presented above, we have assumed a 4% annual discount rate on future costs. As a result, the impact of current costs is greater than the impact of future costs, and costs far in the future are diminished until they no longer matter.

In spite of the impact that a discount rate has on balancing short-term and long-term costs, small changes in the discount rate do not appear to significantly affect control policies or expected costs. Figure 11E5 is a plot of the pumping and recharge policies using a zero discount rate. Compared with the results of Figure 11D4 using a 4% discount rate, we should pump less and recharge more. In addition, the supply policy (not shown) releases less water to users. Overall, the character of the solutions using 4% and 0% discount rates are the same, with marginal differences in the timing of rationing and reallocation.

Figure 11E6 is a plot of the expected annual cost as a function of pumping and recharge capacities using a zero discount rate. This cost does not depend on the initial

conditions since, regardless of initial conditions, expected annual costs converge to the same values with a long enough time horizon and total costs increase linearly with time. For the conjunctive-use problem, we observe that policies change little after a few decades. In effect, Figure 11E6 represents the infinite-horizon, steady-state case.

Using a smaller discount rate, we should ration more frequently and maintain higher storage levels. As a result, we expect to see annual costs that are somewhat higher. Also, we expect to see an increase in the benefit from expanding capacities since the somewhat larger benefit of storing water. However, we see that the expected annual cost using a zero discount rate (Figure 11E6) falls between the favorable and unfavorable costs expected annual cost using a 4% discount rate (Figures 11E2 and 11E4). This is because of the larger impact that initial storage conditions have on expected costs when using a non-zero discount rate.

In addition, there is also a greater value in storing more of this water in the subsurface using a smaller discount rate. Limits on pumping and recharge (combined with larger subsurface storage capacity) mean that it is likely that groundwater will be used to meet long-term demands and surface water will be used to meet short-term demands. With a lower discount rate, the long-term benefits of groundwater increase its value relative to surface water. For maximum development of surface storage capacity, Figure 11E6 shows an annual decrease in cost of \$4.8 million a year (\$17.8 million minus \$13.0 million). For maximum development of groundwater, the decrease is \$8.0 million a year. This compares to annual decreases of \$4.4 million and \$8.3 million in Figure 11E2 and of \$4.4 million and \$5.7 million in Figure 11E4. As expected, the benefit of expanding surface storage capacity is greater with a zero discount rate. The benefit of groundwater development is more difficult to compare because of the influence of initial storage levels; however, the benefit using a zero discount rate is close to the higher value of \$8.3 million.

Figure 11E5. Pumping and Recharge (TAF per year) with a Zero Discount Rate

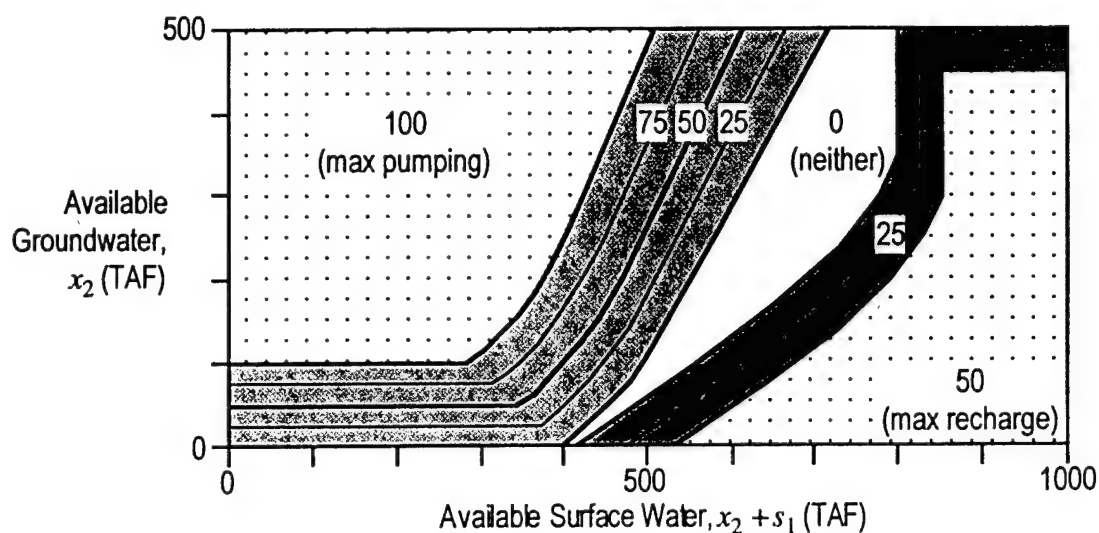
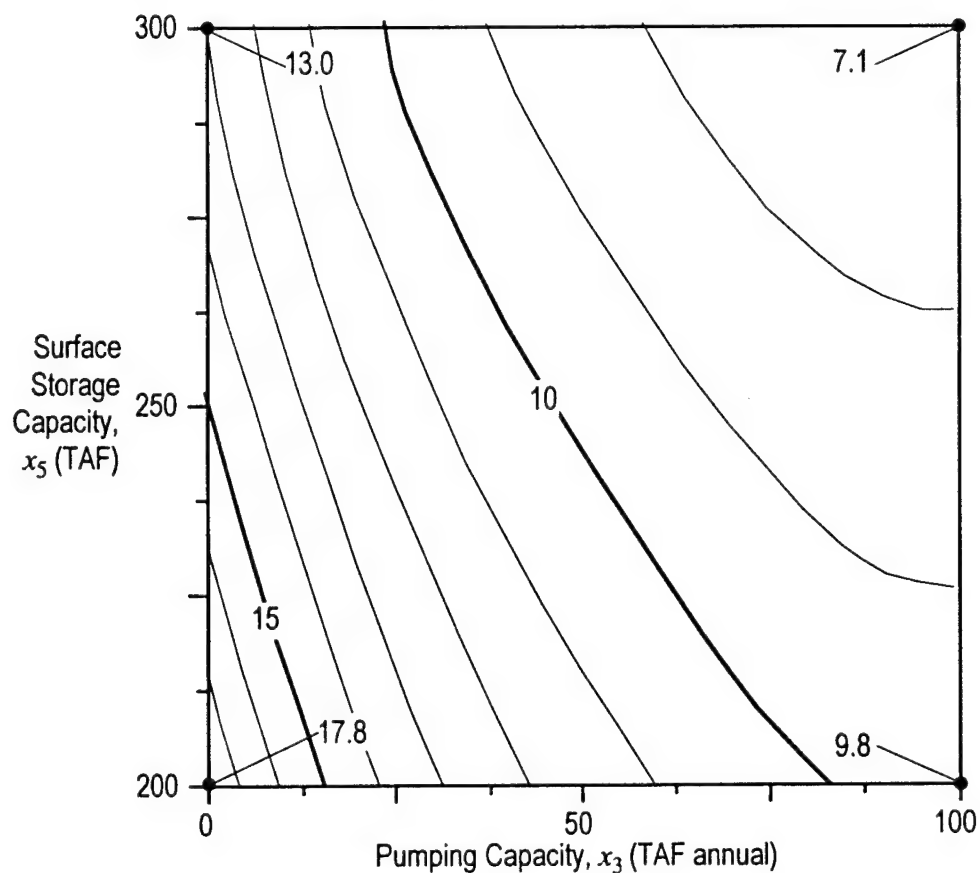


Figure 11E6. Expected Annual Cost (Million \$) for Different Levels of Conjunctive Development with Zero Discount Rate



F. CONCLUDING REMARKS ON THE CONJUNCTIVE MANAGEMENT OF SURFACE AND GROUNDWATER STORAGE

For many systems, including those that conjunctively manage groundwater and surface water, it may be difficult to identify appropriate control policies and capacity-expansion alternatives without using systems analysis techniques to identify the best policies. Systems analysis is especially valuable when variable and uncertain inputs drive the state of a system. In particular, conjunctive-use systems are dynamic systems driven by inputs that often are highly variable and uncertain.

We have developed optimal control policies for a simple conjunctive-use model by applying systems analysis. These policies identify decisions to supply water to users and to allocate stored water between a surface reservoir and an aquifer. Because of the year-long time step and treatment of groundwater as a "bathtub" with only one state variable, the model is too simple for practical application to the real system. However, modification of the simple conjunctive-use model for practical application to the real system is not difficult, and the number of state variables required is within the ability of new DDP methods. Also, the model captures important differences between surface and subsurface storage, and the results do give us some understanding of effective conjunctive-use management, particularly when the water resources of a system are almost fully utilized.

Control policies that result from systems analysis of the simple conjunctive-use model require information on surface and subsurface storage levels and current inflows. Because of the impact of hedging, these control policies cannot be described by simple heuristic rules. Hedging is apparent in decisions that ration current supplies (to balance the current benefits of water use with future benefits) and that allocate supplies between surface and subsurface storage (to maximize water-supply reliability).

We have demonstrated that efficient control of conjunctive-use systems requires decisions that cannot be identified by simple rules but, instead, requires detailed control policies that consider complete state information. When approaching full utilization of water resources in a system (as in the conjunctive-use model), these control policies make active use of pumping and recharge to meet demands and to allocate water between surface and subsurface storage, in spite of associated operating costs. Also, we observe that pumping and recharge should be managed in a manner that may conflict with heuristic or "common sense" control. In particular, we should reduce pumping and increase recharge when groundwater levels are low, even if insufficient surface water is

available to meet all demands and to fill surface reservoirs. Recharge may be appropriate even when rationing water supplies, if groundwater levels are sufficiently low. Though such cautious management is unexpected, we see that cautious management significantly improves system reliability by reducing the expected cost of rationing. However, even when the value of cautious management is recognized, the identification of system conditions that require hedging (i.e., rationing and cautious allocation decisions) can be difficult and contentious. Systems analysis allows us to identify these conditions using an objective approach that produces unambiguous results.

In addition, we have demonstrated that water managers can use systems analysis to evaluate the best mix of facilities used to expand system capabilities. Changes in system configuration, inputs, or goals require identification of new control policies that are suited to the new conditions. We present a systems-analysis approach that allows use to evaluate these changes while simultaneously identifying the best control policies. We accomplish this by augmenting the state information with variables that identify pumping, recharge, and surface-storage capacity. This convenient approach allows us to identify the best mix of capacities. This is a significant improvement over a trial-and-error approach that requires use to iteratively modify system capacities and separately evaluate control policies and an expected benefit for the modifications.

Though a simple case, the conjunctive use model has validated the application of systems analysis to the real-time control and capacity expansion planning. In particular, the application to the simple conjunctive-use model has permitted improved understanding of efficient conjunctive-use operations and plans. We have used the EBMUD system as a case study, though these results have been presented without case-specific complexities. Modifications to the model that would permit solution of specific problems for the EBMUD system include shorter stages and state variables for additional storage reservoirs and parameters to improve streamflow forecasts (to incorporate streamflow autocorrelation and correlation with snowpack measurements). Though these modifications will increase the computational effort required to solve the conjunctive-use problem, they are within the capabilities of the systems analysis methods employed.

In summary, we have used systems analysis to develop real-time controls for a conjunctive-use system. These controls make use of differences between surface water and groundwater to improve water supply reliability. In addition, we have evaluated the benefit capacity expansion alternatives in a dynamic model with five state variables. Based on benefit and cost comparison of these results, we observe that adding subsurface storage to existing surface-reservoir systems can improve water-supply reliability for less than one-tenth the cost of increasing surface storage.

CHAPTER 12.

CONCLUSIONS

Discrete Dynamic Programming (DDP) is a general optimization method that can be used in systems analysis of complex non-linear stochastic control problems. Unfortunately, the application of DDP is limited by the "curse of dimensionality" that prevents its application to systems that must be described by many state variables. Using traditional methods, the application of DDP is restricted to system models that include two or three state variables.

This thesis presents methods that allow DDP to be applied without excessive computational effort to stochastic control problems with as many as six to eight state variables. This is accomplished by Hermite-interpolation methods (Chapter Five) and Gaussian-quadrature methods (Chapter Seven) that are significantly more accurate than traditional methods. The high-order accuracy of these methods permits development of accurate control policies with coarse discretizations of the state variables and stochastic variables, reducing the effort to solve stochastic control problems by several orders of magnitude.

The enhanced ability of DDP permits the solution of stochastic control problems that were previously beyond the ability of systems analysis. In the past, solutions could not be obtained without making assumptions that were correct only if certain conditions held. Frequently, these assumptions required that the condition of certainty equivalence held. At other times, it was assumed that solutions fit pre-determined functional forms. These various assumptions are unnecessary for DDP, and DDP produces the truly optimal solution so long as a system is appropriately modeled by a limited number of state variables.

To demonstrate the ability of the new methods, this thesis applies DDP to several hypothetical problems. In the first set (Chapter Six), DDP is applied to problems with as many as seven state variables. These problems demonstrate the high-order accuracy of Hermite interpolation and the ability to use coarse state discretizations. In the second set (Chapter Eight), DDP is applied to problems with diverse stochastic models. These problems demonstrate the high-order accuracy of Gaussian quadrature and the ability to use coarse discretization of the probability distribution of input variables. In the third set (Chapter Nine), DDP is applied to reservoir problems with autocorrelated inflows and as

many as six state variables. These problems demonstrate that it is important that control policies incorporate an appropriate level of caution to reduce the potential high cost of extreme events. In the final set (Chapter Eleven), DDP is applied to a simple conjunctive-use problem. This problem demonstrates that effective management of groundwater and surface water systems requires complex policies that take advantage of the different characteristics of the surface and subsurface storage.

This final problem also demonstrates our ability to use DDP as a planning tool to evaluate the benefit of capacity expansion alternatives. By adding state variables to represent variable capacity constraints, DDP is used to evaluate the expected benefits for a range of capacity expansion alternatives while simultaneously identifying and applying optimal control policies for each alternative. Thus, DDP can be used to select the best alternative by a rational balancing the capital costs and operating costs. This is but one among many applications of DDP that are now possible using a larger number of state variables.

Building on the example of the conjunctive-use problem, we can anticipate a variety of other applications. Some of these applications that I hope to include in future work include:

(1) *Applications that result in practical use:* The DDP methods of this thesis have practical value that can be demonstrated in application to real-world problems. This includes application of DDP to more detailed models of the EBMUD system or to other systems whose managers support a systems-analysis approach.

(2) *Measuring the value of forecast information in real-time control:* For example, accurate measurements of precipitation can significantly improve predictions of drought and flood, but collection of these measurements comes at a significant cost. DDP can be used to identify cost-effective data collection strategies.

(3) *Balancing competing goals in multipurpose control:* Reservoir management usually requires consideration of multiple and conflicting objectives. DDP can identify control policies that balance conflicting objectives if we are willing to explicitly state the values of these objectives (such as in Chapter Ten where we estimate the value of urban water supplies). Even when these values are approximate, DDP can develop better multipurpose control policies than heuristic methods. Also, this process can be reversed to identify values associated with preferred control policies and may be used to identify policies that use inappropriate values.

(4) *Measuring the local costs of climate change:* The scientific community is quickly coming to a consensus on the large-scale impacts of climate change, and global climate models may provide useful estimates on medium-scale hydrologic trends. DDP is a systems-analysis tool that can provide the necessary connection between these trends and their effect on local water-management. DDP is a useful tool in identifying operations and plans that best respond to changing hydrology, water needs, and management systems.

(5) *Aquifer management modeling:* Hydrogeologic models describe spatially variable quantities that are not easily described by a few state variables. However, additional state variables permitted by the new DDP methods may permit the application of DDP to simple hydrogeologic models. This may be useful, for example, when using systems analysis to identify cautious management policies.

In addition, it seems reasonable to extend the methods and application of the interpolation and quadrature methods of this thesis. These research opportunities may yield further improvement in DDP or may be used to improve other stochastic optimization methods. Some of these extensions include:

(1) *Discretization of cost-to-go functions using adaptive grids:* Adaptive grids can apply fine state discretization only where needed. This can significantly reduce the number of state-space nodes, especially for problems with complex cost-to-go functions. Furthermore, adaptive grids can be used to automate the discretization process.

(2) *Application of Hermite interpolation and Gaussian quadrature to stochastic dual dynamic programming (SDDP):* SDDP discretizes the state space without using a grid by locating discrete states where needed most. This can significantly reduce the number of discrete states required to approximate a cost-to-go function. However, interpolation of the cost-to-go by cutting planes does not produce high-order accuracy, and it may be useful to apply a high-order Hermite interpolation without interfering with the linear solver. Also, Gaussian quadrature may suggest a more efficient method of selecting scenarios.

(3) *Interpolation methods that preserve cost-to-go convexity:* As of yet, no computationally efficient interpolation methods have been proposed that produce smooth function approximations while guaranteeing convexity. Using a finer grid to discretize the function domain tends to overcome problems with convexity, but this comes at a cost of substantially greater computational effort.

(4) *Higher-order interpolation methods:* A logical extension of the second-order Hermite method is to include other high-order derivatives of the cost-to-go. For example, the second-order method could be modified to include all second derivatives of the Hessian, and not just the off-diagonal elements. While this will increase the order of interpolating polynomials, the improved accuracy may offset the additional effort and potential for oscillation. In addition, other non-polynomial functions may be used for interpolation (e.g., complex exponential functions to represent periodic component).

In summary, DDP can be applied to a range of new, more-complex applications. While DDP is but one method for the optimal control of stochastic and dynamic systems, it has unique abilities that make it the preferred optimization method for some applications. This thesis takes a few tentative steps towards these applications.

APPENDIX A.

SUMMARY OF NOTATION

This appendix provides a summary of notation and equations in common use in this thesis.

1. NOTATION

Authors have frequently developed notation adapted to suit their application of systems analysis. To the extent possible, notation in this thesis adheres to that used in existing optimal-control literature [Stengel, 1994] and that used by *Foufoula-Georgiou and Kitanidis* [1988], though I have also taken some liberty to avoid confusion. Tables A1-7 summarizes the notation presented in this thesis.

Table AA1. Notation for System Model

t	time or stage
Δt	stage length
n	number of state variables
m	number of random variables
N	number of stages
\mathbf{u}	vector of decision variables
\mathbf{x}	vector of n state variables
\mathbf{y}	vector of state variables $\mathbf{y}_{(t)} \equiv \mathbf{x}_{(t+\Delta t)}$
\mathbf{s}	vector of stochastic variables
\mathbf{w}	vector of m normally-distributed random variables
$\mathbf{B}_t^L, \mathbf{B}_t^U$	vectors of lower and upper bounds on decision variables
\mathbf{T}_t	vector of functions describing the transition of state $\mathbf{x}_{(t)}$ to $\mathbf{x}_{(t+\Delta t)}$
\mathbf{S}_t	vector of functions modeling stochastic inputs $\mathbf{s}_{(t)}$
W_t	probability density function
\mathbf{Q}_t	covariance matrix of stochastic variables

Table AA2. Conventionally Defined and Non-Specific Parameters, Variables, and Functions

μ	mean
σ	standard deviation
a, b, c, h	arbitrary parameters
f, g	arbitrary function

Table AA3. Notation for Value Model and Optimal Solution

C_t	cost function for current stage
$F_{t_{N+1}}$	cost function for terminal state
V_t	total-cost function
E	expected value operator
\mathbf{u}^*	vector of optimal control decisions
$\mathbf{U}_{(t)}$	vector of control policy functions
$\mathbf{U}_{(t)}^*$	vector of optimal control policy functions
F_t	cost function for expected cost-to-go from state $\mathbf{x}_{(t)}$

Table AA4. Notation for Effort of Discrete Dynamic Programming

J	total computational effort for each stage
Z	total computational effort for each node
Z_l	computational effort for each evaluation of the total-cost function
Z_S	number of evaluations of the total-cost function in each search
Z_0	computer processor speed (seconds per flop)
Z_1	number of searches for each node
Z_2	number of searches to verify solution for each node and outcome
$\mathbf{x}^{(i)}$	discrete state (i) at node of the state-space grid
Λ	number of discrete values per state variable
Λ^n	total number of state-space nodes (i.e., discrete states)
$\mathbf{w}^{(k)}$	discrete outcome (k) at node of the stochastic-space grid
K	number of discrete values per stochastic variable
K^m	total number of stochastic-space nodes (i.e., discrete outcomes)

Table AA5. Notation for Interpolation of the Cost-To-Go Function

$\mathbf{x}^{(\gamma)}$	discrete state at corner node (γ) of current hypercube
F_γ	function value at node $\mathbf{x}^{(\gamma)}$
\mathbf{G}_γ	function gradient at node $\mathbf{x}^{(\gamma)}$, $\mathbf{G}_\gamma = dF/d\mathbf{x}^{(\gamma)} = \partial F/\partial \mathbf{x}^{(\gamma)}$
\mathbf{H}_γ	function Hessian at node $\mathbf{x}^{(\gamma)}$
ϕ_γ	interpolation weight on F_γ
ψ_γ	vector of interpolation weights on \mathbf{G}_γ
χ_γ	matrix of interpolation weights on \mathbf{H}_γ
$\mathbf{x}^{\text{low}}, \mathbf{x}^{\text{high}}$	discrete-state nodes at lowest/highest hyper-cube corners
ξ	location of state \mathbf{x} in the local coordinates of a hypercube
$\eta^{(\gamma)}$	normalized distance of state \mathbf{x} from node $\mathbf{x}^{(\gamma)}$
R, P, Q	arbitrary weighting functions
$\alpha, \beta, \omega, \delta$	shorthand notation for weighting function polynomials

Table AA6. Notation for Numerical Integration of the Expected Cost-To-Go

z	arbitrary stochastic variable
p_K	polynomial of order K whose roots locate Gauss-quadrature abscissas
v_j	weight applied to the j 'th root of p_K
$v_{t,k}$	weight applied to the k 'th outcome of the m random variables $\mathbf{w}_{(t)}^{(k)}$

Table AA7. Notation for Rationing-Cost Function

α	coefficient of demand elasticity (note: duplicate use)
P	price of water (per acre foot) (note: duplicate use)
Q	quantity of water (acre feet) (note: duplicate use)
P^*	price of water that equalizes supply and demand
Q^*	quantity of water supplied or demanded at P^*

2. EQUATIONS

The following is a summary of the mathematical functions used for systems analysis in this thesis. The following functions use the notation of Tables A1-4. Equation numbers correspond to the presentation of these equations earlier in the thesis.

1. System Model

Recursive definitions:

$$\mathbf{y} \equiv \mathbf{x}_{(t_{j+1})}, \quad \mathbf{x} \equiv \mathbf{x}_{(t_j)}, \quad \mathbf{u} \equiv \mathbf{u}_{(t_j)}, \quad \mathbf{w} \equiv \mathbf{w}_{(t_j)}, \quad t \equiv t_j \quad (2C1)$$

$$\text{Transition functions:} \quad \mathbf{y} = \mathbf{T}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2C2)$$

$$\text{Stochastic model:} \quad \mathbf{s} = \mathbf{S}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2C3)$$

$$\text{Constraints:} \quad \mathbf{B}_{(t)}^L(\mathbf{x}, \mathbf{w}) \leq \mathbf{u} \leq \mathbf{B}_{(t)}^U(\mathbf{x}, \mathbf{w}) \quad (2C4)$$

2. Value Model

$$\text{Value function:} \quad V_{t_1} = \sum_{t=t_1}^{t_N} C_t(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{N+1}}(\mathbf{x}) \quad (2C7)$$

3. General Solution

$$\text{Cost-to-go function:} \quad F_{t_1}(\mathbf{x}) = E_{\mathbf{w}_{(t_1)}, \dots, \mathbf{w}_{(t_N)}} \{ V \mid \mathbf{U}_{(t_1)}, \dots, \mathbf{U}_{(t_N)} \} \quad (2C15)$$

$$\text{Control policy:} \quad \mathbf{u}^* = \mathbf{U}_{(t)}^*(\mathbf{x}, \mathbf{w}) \quad (2C16)$$

4. Dynamic Programming Solution

$$\text{Total cost function:} \quad V_{t_j} = C_{t_j}(\mathbf{x}, \mathbf{u}, \mathbf{w}) + F_{t_{j+1}}(\mathbf{y}) \quad (4A5)$$

$$\text{Cost-to-go function:} \quad F_{t_j}(\mathbf{x}) = E_{\mathbf{w}} \{ \min_{\mathbf{u}} \{ V_{t_j} \} \} \quad (4A6)$$

5. Interpolation

$$\text{Multilinear:} \quad \hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_n} \{ \phi_{\gamma}(\mathbf{x}) F_{\gamma} \} \quad (4D1)$$

$$1^{\circ} \text{ Hermite:} \quad \hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_n} \{ \phi_{\gamma}(\mathbf{x}) F_{\gamma} + [\psi_{\gamma}(\mathbf{x})]^T \cdot \mathbf{G}_{\gamma} \} \quad (4D3)$$

2° Hermite:

$$\hat{F}^{(i)}(\mathbf{x}) = \sum_{\gamma=\gamma_1}^{\gamma_n} \{ \phi_{\gamma}(\mathbf{x}) F_{\gamma} + \sum_{j=1}^n \{ \psi_{\gamma,j}(\mathbf{x}) G_{\gamma,j} + \sum_{k=j}^n \{ \chi_{\gamma,j,k}(\mathbf{x}) H_{\gamma,j,k} \} \} \} \quad (5H1)$$

APPENDIX B.

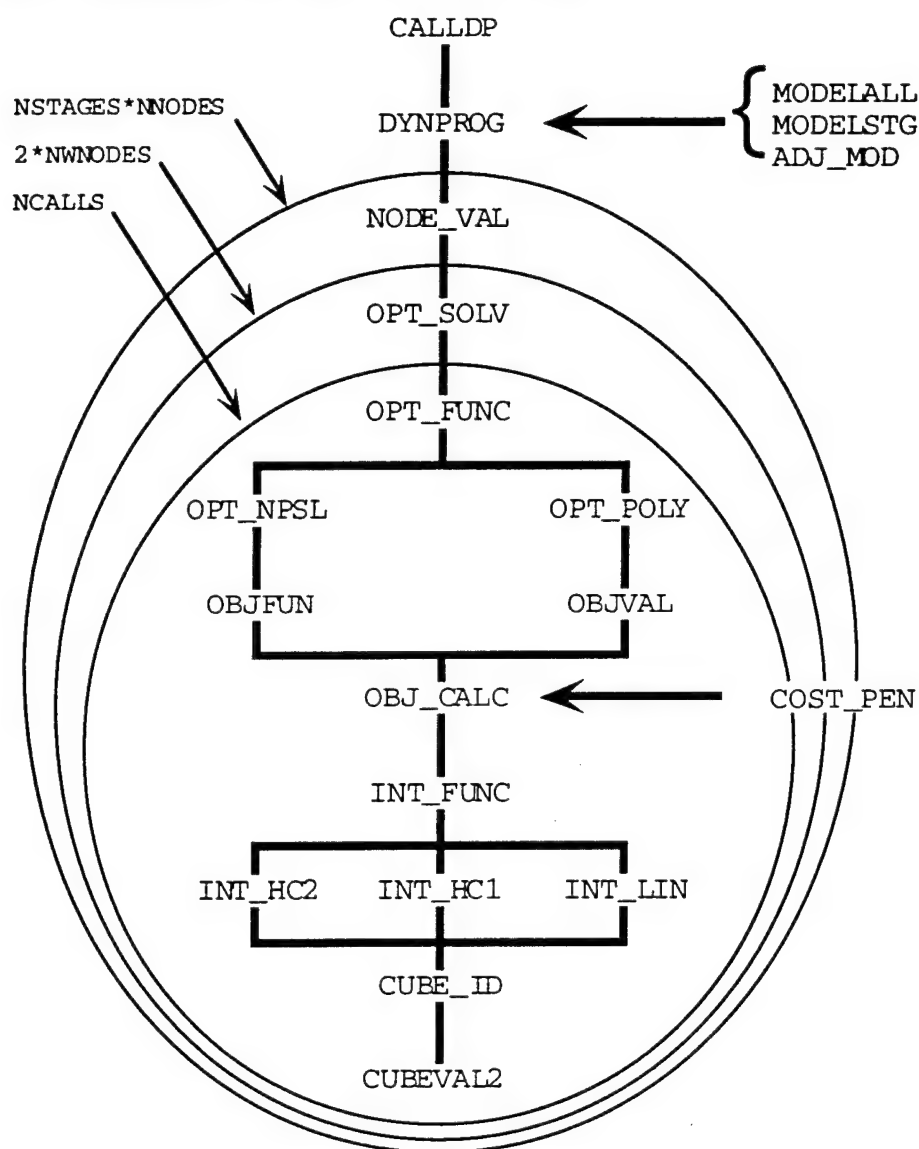
COMPUTER CODE FOR DISCRETE DYNAMIC PROGRAMMING AND ENHANCEMENTS

This appendix provides the code used to implement the discrete dynamic programming methods presented in this thesis. The following code is written in standard FORTRAN 90 with exception for the machine-dependent statements used to track computational time. The compiler did not have a complete implementation of the FORTRAN 90 standard, so the more restrictive FORTRAN 77 standard is used by much of the code.

1. SIMPLIFIED FLOW CHART

Figure A1 provides an overview of heirarchy of the DDP routines. The ovals indicate that routines are inside a loop, with the number of loops indicated to the top left. Not included are the user-supplied routines that define a systems-analysis problem. These routines feed routines used to model and solve the problem.

Figure AB1. Simplified Flow Chart for the DDP code



2. INCLUDE FILES FOR COMMON STORAGE OF DATA

Include files are used throughout the code to ensure uniform specification of variables and to allow control over the allocation of storage.

Include File I.SIZEALLO

This include file identifies parameters used to allocate memory in common arrays. The parameters **MAXNODES** and **MAXWNODES** allocate memory to store nodes of the state-space and stochastic-space grids. The numbers of nodes can be large with fine

discretization and high dimension, and may exceed available memory of a computer. The maximum numbers of state variables MAXNX and stochastic variables MAXNW also significantly influence required memory.

```
!-----
!Parameters to allocate storage space:
! MAXNX    = max # state variables (dimensions)
! maxidx   = max # discrete values in any one state variable
! maxnodes = max # nodes identifying discrete states of cost function
! MAXNW    = max # stochastic variables
! maxwnodes= max # discrete realizations of the stochastic variables
!-----
```

```

      INTEGER          MAXNU,   MAXNX,   MAXIDX,
+
+                     MAXNW, MAXIDW, MAXCON,
+                     MAXSEAS, MAXSTAGES,
+                     MAXWNODES, MAXNODES, MAXCORN
      PARAMETER        ( MAXNU   = 4,   MAXNX   = 4,   MAXIDX   = 17 )
      PARAMETER        ( MAXNW   = 2,   MAXIDW  = 9,   MAXCON   = 10 )
      PARAMETER        ( MAXSEAS = 12,         MAXSTAGES= 50 )
      PARAMETER        ( MAXNODES = 2000, MAXWNODES= 100 )
      PARAMETER        ( MAXCORN  = 256 )

```

Include File I.SIZEPROB

This include file identifies common variables that specify the actual size and other characteristics of a discrete dynamic programming problem. These values are used in many routines to allocate memory to variables.

```

      INTEGER          NU, NX, NNODES, NWNODES, NW,
+
+                     NSTAGES, NSEAS,
+                     NTNLN, NCNLN,
+                     NBASE2
      DOUBLE PRECISION DISCOUNT, TIGHT, FTOL, UTOL
      LOGICAL          GDP, NEWTON, STOCHASTIC
      COMMON /SIZEPROB/ NU, NX, NNODES, NWNODES, NW,
+
+                     NSTAGES, NSEAS,
+                     NTNLN, NCNLN,
+                     NBASE2,
+                     DISCOUNT, TIGHT, FTOL, UTOL,
+                     GDP, NEWTON, STOCHASTIC

```

Include File I.XNODES

This include file identifies common arrays that store the location for each state-space node in array XN. Array IABOVE and IBELOW are used as pointers to nodes that are immediately above and below.

```

      INTEGER          IABOVE (MAXNX, MAXNODES) ,
+
+                     IBELOW (MAXNX, MAXNODES)
      DOUBLE PRECISION XN (MAXNX, MAXNODES)

```

```
COMMON /XNODES/      IABOVE, IBELOW, XN
```

Include File I.SPECW

This include file identifies common arrays that store the location of each stochastic-space node (i.e., each outcome) in array WN. Array PWN stores the weight of each outcome and array LIKELY is used to identify whether an outcome is sufficiently likely to be used in evaluating expected values.

```
DOUBLE PRECISION      WN(MAXNW,MAXWNODES), PWN(MAXWNODES)
LOGICAL               LIKELY(MAXWNODES)
COMMON /SPECW/        WN, PWN, LIKELY
```

Include File I.FNODES

For each state-space node, this include file identifies common arrays that store the cost-to-go (array FN) and first derivatives (array FXN).

```
DOUBLE PRECISION      FN(MAXNODES), FXN(MAXNX,MAXNODES)
COMMON /FNODES/       FN, FXN
```

Include File I.CONTROL

This include file identifies common arrays that store the linear constraints and bounds on control variables U and state variables X. Note that the code is not yet adapted to allow non-linear constraints.

```
INTEGER               NTLIN, NCLIN, NLCON
DOUBLE PRECISION      UBL(MAXNU), UBU(MAXNU),
+                     UGUESS(MAXNU), USCALE(MAXNU),
+                     XBL(MAXNX), XBU(MAXNX),
+                     YBL(MAXNX), YBU(MAXNX),
+                     ACLBL(MAXCON), ACLBU(MAXCON),
+                     ACL(MAXCON,MAXNU+MAXNX+MAXNW),
+                     ABL(MAXCON), ABU(MAXCON),
+                     AA(MAXCON,MAXNU+MAXNX+MAXNW),
+                     X0(MAXNX), W0(MAXNW)
COMMON /CONTROL/      NTLIN, NCLIN, NLCON,
+                     UBL, UBU, UGUESS, USCALE,
+                     XBL, XBU, YBL, YBU,
+                     ACLBL, ACLBU, ACL,
+                     ABL, ABU, AA,
+                     X0, W0
```

```
!badj gives adjusted bounds on linear constraints with initial state
! variables held constant. Generally, badj bounds will be used,
! except when gradients can be obtained directly from solver.
```

Include File I.SPECNOW

This include file identifies common variables that store the current stage of a problem.

```
INTEGER          ISTAGE, IYEAR, ISEASON, IFIRST, IYFIRST, ISW
COMMON /SPECNOW/ ISTAGE, IYEAR, ISEASON, IFIRST, IYFIRST, ISW
```

Include File I.CUBE

This include file identifies common arrays that store hypercube values. These common arrays are used in interpolation subroutines to avoid passing data.

```
INTEGER          LEVEL, IXSTART, IXID(MAXCORN)
DOUBLE PRECISION X(MAXNX), XLO(MAXNX), XHI(MAXNX),
+               DXOUT(MAXNX),
+               FC(MAXCORN), FXC(MAXNX,MAXCORN),
+               FXXC(MAXNX,MAXNX,MAXCORN)
LOGICAL          XOUT
COMMON /CUBE/    LEVEL, IXSTART, IXID, X, XLO, XHI,
+               XOUT, DXOUT,
+               FC, FXC, FXXC
```

Include File I.PERFORM

This include file identifies common variables that store measures used to evaluate performance of the code.

```
INTEGER          LPRINT, N_INT, N_OBJ, N_OPT, N_SOL
DOUBLE PRECISION T_ID, T_VAL, T_WEIGH, T_INT, T_CALL, T_OBJ,
+               T_OPT, T_SOL
LOGICAL          PP
COMMON /PERFORM/ LPRINT, N_INT, N_OBJ, N_OPT, N_SOL,
+               T_ID, T_VAL, T_WEIGH, T_INT, T_CALL, T_OBJ,
+               T_OPT, T_SOL,
+               PP
```

3. INTERPOLATION SUBROUTINES

Interpolated values of the cost-to-go function are evaluated in one of three subroutines that perform multilinear interpolation, first-order Hermite interpolation, or second-order Hermite interpolation. The subroutine INT_FUNC selects interpolation subroutine INT_LIN, or INT_HC1, or INT_HC2, as appropriate. These subroutines also call the subroutines CUBE_ID and CUBEVAL2 to identify the the hypercube and node values used in interpolation.

Subroutine INT_FUNC

This subroutine selects the interpolation to be used. It is provided as a separate routine to allow convenient insertion of other interpolation routines.

```
SUBROUTINE INT_FUNC ( LEVEL, X,
+                   F, FX )

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Parameters for problem size.
  INTEGER                  LEVEL
  DOUBLE PRECISION          X(NX)

  DOUBLE PRECISION          F, FX(NX)

  !-----
  !Specifies interpolation function used to approximate the cost function.
  !-----
  !On input, LEVEL identifies what derivatives are interpolated:
  ! 0: only f
  ! 1: f and fx
  !On output, if LEVEL changed to zero, this is used as a flag to indicate
  ! that Gradient DP will not be used (for example, if linear
  ! interpolation is used to estimate the cost function).
  !-----

  INTEGER                  HERMORD

  EXTERNAL                  INT_LIN, INT_HC1, INT_HC2

  !-----
  !
  !                               -----
  !                               Specify order of Hermite interpolation.
  !                               -----
  !

  HERMORD = 2

  !
  !                               -----
  !                               Call Interpolation routine.
  !                               -----
  !

  IF (GDP) THEN
    IF (HERMORD.EQ.1) THEN
      CALL INT_HC1 (LEVEL,X, F,FX)
    ELSEIF (HERMORD.EQ.2) THEN
      CALL INT_HC2 (LEVEL,X, F,FX)
    ELSE
      WRITE (*,*) '(INT_FUNC) ERROR:  HERMORD =', HERMORD
      STOP 'INT_FUNC'
    END IF
  ELSE
    CALL INT_LIN (LEVEL,X, F,FX)
  END IF

END
```

Subroutines INT_HC2, INT_HC1, and INT_LIN

This subroutine uses the second-order Hermite interpolation method to evaluate interpolated values and derivatives. These are evaluated as a weighted sum of the discrete values and derivatives at corner nodes of the surrounding hypercube. The nodes of the hypercube are identified by the subroutine CUBE_ID and the corner node values are identified by the subroutine CUBEVAL2. Common storage is used to avoid passing data between subroutines. The effort to evaluate interpolated values is divided into two parts: (1) evaluation of weights, and (2) application of the weights to the sum for each interpolant.

The subroutine INT_HC1 is the same as INT_HC2 except that second derivatives are not used. Subroutine INT_HC1 does not include portions of the code that evaluates and applies second-derivatives and weights. Subroutine INT_LIN is also structure similar to INT_HC2 except that derivatives are not used and the weighting functions are n -fold linear functions.

```
SUBROUTINE INT_HC2 ( LEVELINT, XINT,
+                  F, FX )

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
  INTEGER                  LEVELINT
  DOUBLE PRECISION          XINT(NX)

  DOUBLE PRECISION          F, FX(NX)

!-----
!Determines interpolated function value and derivatives
! using a hermite interpolation that produces continuous 1st derivatives
!-----
!LEVEL identifies if derivatives are to be calculated:
! 0: only f
! 1: f and fx
! 2: f, fx, and fxx ****not yet
!-----

  INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
  INCLUDE 'I.PERFORM'       !Track performance of solver and output.
  INCLUDE 'I.CUBE'          !Holds corner node values for interp.

!-----

!              Other local variables.

  INTEGER                  LEVEL0, J, J1, J2, K, K1, IBASE2(0:NX),
+                          I, INEW, IP
  DOUBLE PRECISION          TOL, DX(NX), DXINV(NX), XI(NX),
+                          ETA1(NX), ETA2(NX),
+                          EE1(NX), EE2(NX),
+                          C1(NX), C2(NX), D1(NX), D2(NX),
```

```

+          E1(NX), E2(NX),
+          B1(NX), B2(NX), Q(NX),
+          A1(NX), A2(NX),
+          PHI(NBASE2), PSI(NX,NBASE2),
+          CHI(NX,NX,NBASE2),
+          DPHI(NX,NBASE2), DPSI(NX,NX,NBASE2),
+          DCHI(NX,NX,NX,NBASE2)
LOGICAL      P
REAL        SECNDS, TIME0, TIME1

EXTERNAL    CUBE_ID, CUBEVAL2

!-----
      N_INT = N_INT + 1
      TIME0 = SECNDS(0.0)

!-----
!      Verify inputs.
!-----

IF ( (LEVELINT.LT.0).OR.(LEVELINT.GT.1) ) THEN
  WRITE (*,*) '(INT_HC2) LEVEL INCORRECT,', LEVELINT
  STOP
END IF

!-----
!      Initialize values.
!-----

TOL = 1.0E-14
LEVEL = LEVELINT
X(:NX) = XINT
LEVEL0 = LEVEL

IBASE2(0) = 1
DO J = 1,NX
  IBASE2(J) = IBASE2(J-1)*2
END DO

!-----
!      Identify hypercube.
!-----

CALL CUBE_ID
IF ( (XOUT).AND.(LEVEL.EQ.0) ) LEVEL = 1

!-----
!      Get value at hypercube corner nodes.
!-----

CALL CUBEVAL2

!-----
!      Identify standardized x in the local coordinate
system.
!-----

```

```

TIME1 = SECNDS(0.0)

DX = XHI(:NX) - XLO(:NX)
DXINV = 1.0/DX
XI = (X(:NX) - XLO(:NX))*DXINV

```

```

! -----
! Define convenient variables.
! -----

```

```

ETA1 = XI
ETA2 = 1.0 - XI
EE1 = ETA1*ETA1
EE2 = ETA2*ETA2
A1 = (1.0 + ETA1 + ETA1)*EE2
A2 = (1.0 + ETA2 + ETA2)*EE1
B1 = ETA1*EE2*DX
B2 = -ETA2*EE1*DX

IF (LEVEL.GT.0) THEN
  C2 = 6.0*ETA1*ETA2*DXINV
  C1 = - C2
  D1 = (ETA2 - ETA1 - ETA1)*ETA2
  D2 = (ETA1 - ETA2 - ETA2)*ETA1
END IF

```

```

! -----
! Calculate weights for each node.
! -----

```

```

! Calculate weights recursively (1-D at a time) to
! reduce work by using values calculated in previous
! recursion. Must calculate values for nodes added
! in a recursion before updating values for nodes
! already added.

```

```

DPSI(1,1,2) = D2(1)
DPHI(1,2) = C2(1)

```

```

DPSI(1,1,1) = D1(1)
DPHI(1,1) = C1(1)

```

```

PSI(1,2) = B2(1)
PHI(2) = A2(1)

```

```

PSI(1,1) = B1(1)
PHI(1) = A1(1)

```

```

DO J = 2,NX
  J1 = J - 1
  J2 = J - 2
  IP = IBASE2(J1)
  DO I = 1,IP
    INEW = I + IP

```

```

    IF (LEVEL.GE.1) THEN

```

```

! Derivative weights on second-derivative values.

```



```

DO K = 2,J1
  K1 = K - 1
  DCHI(:J1,:K1,K,INEW) = DCHI(:J1,:K1,K,I)*A2(J)
  DCHI( J,:K1,K,INEW) = CHI(:K1,K,I)*C2(J)
  DCHI(:J1,:K1,K,I) = DCHI(:J1,:K1,K,I)*A1(J)
  DCHI( J,:K1,K,I) = CHI(:K1,K,I)*C1(J)
END DO
DCHI(:J1,:J1,J,INEW) = DPSI(:J1,:J1,I)*B2(J)
DCHI( J,:J1,J,INEW) = PSI(:J1,I)*D2(J)
DCHI(:J1,:J1,J,I) = DPSI(:J1,:J1,I)*B1(J)
DCHI( J,:J1,J,I) = PSI(:J1,I)*D1(J)

!           Derivative weights on first-derivative values.

DPSI(:J1,:J1,INEW) = DPSI(:J1,:J1,I)*A2(J)
DPSI( J,:J1,INEW) = PSI(:J1,I)*C2(J)
DPSI(:J1, J,INEW) = DPHI(:J1,I)*B2(J)
DPSI( J, J,INEW) = PHI(I)*D2(J)
DPSI(:J1,:J1,I) = DPSI(:J1,:J1,I)*A1(J)
DPSI( J,:J1,I) = PSI(:J1,I)*C1(J)
DPSI(:J1, J,I) = DPHI(:J1,I)*B1(J)
DPSI( J, J,I) = PHI(I)*D1(J)

!           Derivative weights on function values.

DPHI(:J1,INEW) = DPHI(:J1,I)*A2(J)
DPHI( J,INEW) = PHI(I)*C2(J)
DPHI(:J1,I) = DPHI(:J1,I)*A1(J)
DPHI( J,I) = PHI(I)*C1(J)

END IF

!           Value weights on second-derivative values.

DO K = 2,J1
  K1 = K - 1
  CHI(:K1,K,INEW) = CHI(:K1,K,I)*A2(J)
  CHI(:K1,K,I) = CHI(:K1,K,I)*A1(J)
END DO
CHI(:J1,J,INEW) = PSI(:J1,I)*B2(J)
CHI(:J1,J,I) = PSI(:J1,I)*B1(J)

!           Value weights on first-derivative values.

PSI(:J1,INEW) = PSI(:J1,I)*A2(J)
PSI( J,INEW) = PHI(I)*B2(J)
PSI(:J1,I) = PSI(:J1,I)*A1(J)
PSI( J,I) = PHI(I)*B1(J)

!           Value weights on function values.

PHI(INEW) = PHI(I)*A2(J)
PHI(I) = PHI(I)*A1(J)

END DO
END DO

```

```

! -----
! Calculate interpolated value.
! -----

```

```

F = 0.0
DO I = 1, IBASE2(NX)
  F = F + PHI(I)*FC(I)
  DO J = 1, NX
    F = F + PSI(J,I)*FXC(J,I)
    DO K = 1, J-1
      F = F + CHI(K,J,I)*FXXC(K,J,I)
    END DO
  END DO
END DO

```

```

! -----
! Calculate interpolated 1st derivatives.
! -----

```

```

IF (LEVEL.GE.1) THEN

  FX = 0.0
  DO I = 1, IBASE2(NX)
    FX = FX + DPHI(:,I)*FC(I)
    DO J = 1, NX
      FX = FX + DPSI(:,J,I)*FXC(J,I)
      DO K = 1, J-1
        FX = FX + DCHI(:,K,J,I)*FXXC(K,J,I)
      END DO
    END DO
  END DO
END IF

```

```

      T_WEIGH = T_WEIGH + SECNDS(TIME1)

```

```

! -----
! Calculate interpolated 2nd derivatives.
! -----

```

```

IF (LEVEL.GE.2) THEN

  WRITE (*,*) '(INT_HC2) ROUTINE NOT ADAPTED'
  STOP

```

```

END IF

```

```

! -----
! Adjust if x outside the domain.
! -----

```

```

IF (XOUT) THEN
  DO J = 1, NX
    F = F + FX(J)*DXOUT(J)
  END DO
END IF

X(:NX) = X(:NX) + DXOUT(:NX)

```

```

DO J = 1,NX
  IF (ABS(X(J)-XINT(J)).GT.TOL) THEN
    WRITE (*,*) '(INT_HC2) ERROR IN X', J, X(J), XINT(J)
    STOP '(INT_HC2)'
  END IF
END DO
LEVEL = LEVEL0

```

```

! -----
! Verify outputs.
! -----

```

```

T_INT = T_INT + SECNDS(TIME0)

```

```

END

```

Subroutine CUBE_ID

This subroutine searches the state-space domain to identify the hypercube that contains the current state. The state-space domain is defined by the grid of nodes stored in common (in the include file I.XNODES). The corner nodes of the identified hypercube are stored in the common (in include file I.CUBE). If the current state is outside the bounds of the discretized state space, the distance away from the closest hypercube is identified for extrapolation of the cost-to-go and derivatives.

```

SUBROUTINE CUBE_ID

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Problem size parameters

!-----
!Identifies corner nodes of hypercube bounding x.
!If x is outside domain, identifies nodes of closest hypercube and
! adjusts x to the closest location at the boundary of the domain.
! The original x is identified by flag xout distance dxout.
!-----

  INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
  INCLUDE 'I.XNODES'        !State discretization.
  INCLUDE 'I.PERFORM'       !Track performance of solver and output.
  INCLUDE 'I.CUBE'          !Holds corner node values for interpolat.

!-----

!
!      Local variables to identify hypercube.

  INTEGER                IBASE2(0:NX), J, IX, IXLO, I, ID(NX),
+                        IC, IXB
  LOGICAL                P

!
!      Other local variables.

  REAL                   SECNDS, TIME0

```

```

!-----
!
!           -----
!           Initialize values.
!           -----
!
!           TIME0 = SECNDS(0.0)
!
!           IBASE2(0) = 1
!           DO J = 1,NX
!             IBASE2(J) = IBASE2(J-1)*2
!           END DO
!
!           XOUT = .FALSE.
!           DXOUT = 0.0
!
!           -----
!           Identify low corner node ixlo of the hypercube that
!           contains x or is closest to x.
!           -----
!
!           Let ixstart be set to previously identified cube;
!           often this will be close to current cube low
!           corner.
!
!           IF ( (IXSTART.LE.0).OR.(IXSTART.GT.NNODES) ) IXSTART = 1
!           IX = IXSTART
!           IF ( (IX.GT.NNODES).OR.(IX.LT.1) ) IX = 1
!
!           For a regular grid, ixlo is the highest node
!           below x. For an irregular (i.e., adaptive grid),
!           this node is the highest node below x that is a
!           low-corner node and whose associated high-corner
!           node is above x.
!
!           DO J = 1,NX
!
!           Move up if current node below x.
!
!           IF (XN(J,IX).LT.X(J)) THEN
!             DO
!               I = IABOVE(J,IX)
!               IF (I.EQ.0) THEN
!                 XOUT = .TRUE.
!                 DXOUT(J) = X(J) - XN(J,IX)
!                 X(J) = XN(J,IX)
!                 IX = IBELOW(J,IX)
!                 IF (IX.EQ.0) THEN
!                   WRITE (*,*) '(CUBE_ID) POINTERS ABOVE AND BELOW = 0', IX
!                   STOP
!                 END IF
!               EXIT
!             ELSE IF (I.EQ.-1) THEN
!               write (*,*) '(cube_id) routine not adapted (1)'
!               stop
!             ELSE IF (XN(J,I).GE.X(J)) THEN
!               EXIT
!             ELSE

```

```

        IX = I
      END IF
    END DO

!           Move down if current node above x.

    ELSE IF (XN(J,IX).GT.X(J)) THEN
      DO
        I = IBELOW(J,IX)
        IF (I.EQ.0) THEN
          XOUT = .TRUE.
          DXOUT(J) = X(J) - XN(J,IX)
          X(J) = XN(J,IX)
          EXIT
        ELSE IF (I.EQ.-1) THEN
          write (*,*) '(cube_id) routine not adapted (2)'
          stop
        ELSE IF (XN(J,I).LE.X(J)) THEN
          IX = I
          EXIT
        ELSE
          IX = I
        END IF
      END DO

!           Current node same as x.

    ELSE
      IF (IABOVE(J,IX).LE.0) THEN
        IX = IBELOW(J,IX)
        IF (IX.EQ.0) THEN
          WRITE (*,*) '(CUBE_ID) POINTERS ABOVE AND BELOW = 0', IX
          STOP
        END IF
      END IF
    END IF

  END DO
  IXLO = IX
  IXSTART = IX

!           -----
!           Identify corner nodes.
!           -----

!           Since node for lower corner is identified, nodes for
!           all other corners can be identified by pointers
!           from previously-identified nodes at lower corners.

  IXID(:NBASE2) = -1
  IXID(1) = IXLO
  ID = 0
  DO IC = 2, IBASE2(NX)

!           Identify current corner.

    DO J = 1, NX
      ID(J) = ID(J) + 1
    
```

```

        IF (ID(J).GT.1) THEN
            ID(J) = 0
        ELSE
            EXIT
        END IF
    END DO

!           Identify associated node.

    DO J = 1,NX
        IF (ID(J).EQ.1) THEN
            IXB = IC - IBASE2(J-1)
            EXIT
        END IF
    END DO
    IX = IABOVE(J,IXID(IXB))

    IXID(IC) = IX

END DO

!           -----
!           Bound hypercube.
!           -----

XLO(:NX) = XN(:NX,IXID(1))
XHI(:NX) = XN(:NX,IXID(NBASE2))

!           -----
!           Verify.
!           -----

T_ID = T_ID + SECNDS(TIME0)

END

```

Subroutine CUBEVAL2

This subroutine uses the corner nodes identified by CUBE_ID (include file I.CUBE) to get the cost-to-go and first derivatives used in interpolation (include file I.FNODES). Second derivatives are calculated by finite differences using the first derivatives and dimensions of the hypercube (include file I.XNODES)

```

SUBROUTINE CUBEVAL2

    IMPLICIT NONE
    INCLUDE 'I.SIZEPROB'      !Problem size parameters

!-----
!Identifies corner-node values, first derivatives, and partial second
!derivatives of hypercube bounding x.
!-----

    INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
    INCLUDE 'I.XNODES'        !State discretization.

```

```

        INCLUDE 'I.FNODES'           !Future cost function.
        INCLUDE 'I.PERFORM'         !Track performance of solver and output.
        INCLUDE 'I.CUBE'           !Holds corner node values for interpolat.

!-----
!
!               local variables.

        INTEGER          J, IC, ID(NX), IX, IX1, IX2, IX3, NODE, N
        DOUBLE PRECISION DX(NX), DELA, DELB, DENOM, W1, W2, W3
        LOGICAL          LININT, P
        REAL             SECNDS, TIME0

!-----
!
!               -----
!               Initialize values.
!               -----
!
!               TIME0 = SECNDS(0.0)
!
!               -----
!               Bound hypercube.
!               -----
!
        DX = XHI(:NX) - XLO(:NX)

!               -----
!               Identify corner node values.
!               -----
!
        ID = 0
        ID(1) = -1
        DO IC = 1, NBASE2
!-----
            IX = IXID(IC)

!               Get value and gradient.

            FC(IC) = FN(IX)
            FXC(:NX, IC) = FXN(:NX, IX)

!               Identify current corner.

            DO J = 1, NX
                ID(J) = ID(J) + 1
                IF (ID(J).GT.1) THEN
                    ID(J) = 0
                ELSE
                    EXIT
                END IF
            END DO

            DO J = 1, NX

!               Identify intervals for calculating Hessian.
!               Note: could reduce effort by over half:
!               (1) Hessian should be symmetric,

```

!

(2) Diagonal elements not needed.

```

IX3 = IABOVE(J,IX)
IX1 = IBELOW(J,IX)
LININT = .FALSE.                                !If true, only 2 nodes
                                                ! available in direction j.

IF (ID(J).EQ.0) THEN
  IF (IX1.GT.0) THEN
    IX2 = IX
    NODE = 2
    DELA = XN(J,IX2) - XN(J,IX1)
    DELB = DX(J)
  ELSE
    IX1 = IX
    IX2 = IX3
    IX3 = IABOVE(J,IX3)
    IF (IX3.GT.0) THEN
      NODE = 1
      DELA = DX(J)
      DELB = XN(J,IX3) - XN(J,IX2)
    ELSE
      LININT = .TRUE.
      IX3 = IX2
    END IF
  END IF
ELSE
  IF (IX3.GT.0) THEN
    IX2 = IX
    NODE = 2
    DELA = DX(J)
    DELB = XN(J,IX3) - XN(J,IX2)
  ELSE
    IX3 = IX
    IX2 = IX1
    IX1 = IBELOW(J,IX1)
    IF (IX1.GT.0) THEN
      NODE = 3
      DELA = XN(J,IX2) - XN(J,IX1)
      DELB = DX(J)
    ELSE
      LININT = .TRUE.
      IX1 = IX2
    END IF
  END IF
END IF

```

!

Calculate finite difference estimate of Hessian.

```

N = NX
IF (LININT) THEN
  DENOM = 1.0/DX(J)
  FXXC(:N,J,IC) = (FXN(:N,IX3) - FXN(:N,IX1))*DENOM
ELSE
  DENOM = DELA*(DELA+DELB)*DELB
  DENOM = 1.0/DENOM
  SELECT CASE (NODE)
  CASE (1)
    W1 = - DELB*(2.0*DELA + DELB)

```



```

IMPLICIT NONE
INCLUDE 'I.SIZEPROB'      !Parameters for problem size.
INTEGER                   LEVEL

DOUBLE PRECISION          U(NU), F, FX(NX)
LOGICAL                   ERROR

!-----
!Specifies solver to find optimal controls and resulting cost f(x)
! and (if requested) gradient fx = df/dx.
!-----
!On input, if ERROR = true on input, diagnostic output is to be provided
! by solver.
!On output, if ERROR = true, there was a problem in solver.
!Solver is applied first to obtain optimum control decisions (with at
! least one restart), and then to obtain objective gradient w.r.t.
! state variables. This approach avoids extra numerical calculation of
! gradients while searching for optimal control decisions.
!On input, LEVEL identifies what derivatives are to be calculated
! directly by solver (if it has the capability):
! 0: only u and f
! 1: u, f, and fx
!On output, LEVEL identifies what derivative were actually calculated.
!U on input is used as starting point for solver.
!If GOODGRAD true, gradients by newton solver as assumed good enough,
! even when there was a problem.
!-----

INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
INCLUDE 'I.CONTROL'       !Constraints on control.
INCLUDE 'I.PERFORM'       !Track performance of solver and output.

!-----

!                               Other local variables.

INTEGER                   LEVEL0, K, NI, MAXITER,
+                         NT_OPT, NT_OBJ, NCHECK
DOUBLE PRECISION          DIFFMAX, USCALE0(NU),
+                         USTART(NU), FSTART, DIFF
LOGICAL                   ERROR0, NEWTON0, DONE, SMALL, BADGRAD
REAL*4                    SECNDS, TIME0

EXTERNAL                  OPT_FUNC

!-----

!                               Identify # restarts used to verify a good solution.

MAXITER = 20

!                               -----
!                               Verify inputs.
!                               -----

IF ( (LEVEL.LT.0).OR.(LEVEL.GT.1) ) THEN
  WRITE (*,*) '(OPT_SOLV) INCONSISTENT LEVEL = ', LEVEL
  STOP '(OPT_SOLV)'

```

```

END IF

!-----
! Save initial settings.
!-----

NT_OBJ = N_OBJ
NT_OPT = N_OPT
TIME0 = SECNDS(0.0)

ERROR0 = ERROR
NEWTON0 = NEWTON
LEVEL0 = LEVEL
USCALE0 = USCALE(:NU)

!-----
!Get optimal control decisions.
!-----

! Call solver as required to get optimal solution.
! Solution will be confirmed by restart.
! If discrepancy occurs, solver called as needed until
! solution converges or max iterations reached.
! Solution of u is used to initialize solver for
! subsequent calls.

LEVEL = 0
NI = 0
USTART = U
DO
!-----
NI = NI + 1

!-----
! Call solver.
!-----

ERROR = ERROR0
IF (NI.GT.MAXITER-3) THEN
  PP = .TRUE.
  ERROR = .TRUE.
END IF
NCHECK = N_OBJ

CALL OPT_FUNC (LEVEL, U,F,FX,ERROR)

!-----
! Specify desired solution accuracy.
!-----

! tolerance is 10x accuracy of npsol based on ftol.

IF (NI.EQ.1) THEN
  IF (F.GT.1.0) THEN
    DIFFMAX = ABS (10.0*F*FTOL)
  ELSE
    DIFFMAX = 10.0*FTOL
  END IF

```

```

END IF

! -----
! If no problem with newton, solution not verified.
! If problem, polytope solver used instead.
! -----

IF (NEWTON) THEN
  IF (.NOT.ERROR) THEN
    EXIT
  ELSE
    NEWTON = .FALSE.

!           Though newton error, solution is propably close.

    USCALE = FTOL
  END IF
END IF

! -----
! Assess convergence of solution.
! -----

DONE = .TRUE.
IF (NI.EQ.1) THEN
  DONE = .FALSE.
  SMALL = .FALSE.
ELSE
  DIFF = ABS( FSTART - F )
  IF (DIFF.GE.DIFFMAX) DONE = .FALSE.

!           Check that number of objective calls indicate that
!           initial polytope was not too small.

  NCHECK = N_OBJ - NCHECK
  IF (NCHECK.LE.NX+NX+2) THEN
    SMALL = .TRUE.
    DONE = .FALSE.
  ELSE
    SMALL = .FALSE.
  END IF
END IF

! -----
! Do loop exit.
! -----

!           If half of maxiter exceeded for newton method,
!           remaining restarts use polytope solver.

IF (DONE) THEN
  BADGRAD = .FALSE.
  DO K = 1,NU
    DIFF = ABS( U(K) - USTART(K) )
    IF (DIFF.GE.UTOL) THEN
      BADGRAD = .TRUE.
      WRITE (*, '(A53,E10.2,I6,20F8.2)')
+      '(OPT_SOLV) WARNING: U NOT STABLE FOR DIFF,K,X,W:',

```

```

+      DIFF,K,X0(:NX),W0(:NW)
      END IF
    END DO
    EXIT
  ELSE
    IF (NI.GE.MAXITER) THEN
      WRITE (*,'(A53,E10.2,6X,20F8.2)')
+      '(OPT_SOLV) ERROR: F NOT STABLE FOR DIFF,X,W: ',
+      DIFF,X0(:NX),W0(:NW)
      EXIT
    END IF

!      Adjust size of polytope for recursive soln.
!      Using small polytope will help convergence; however,
!      polytope must be sufficiently large to ensure
!      that all points are not within ftol. A small
!      number of objective calls indicates that all points
!      were within ftol.

    IF (SMALL) THEN
      USCALE = 100.0*USCALE
    ELSE
      DO K = 1,NU
        IF (NI.EQ.1) THEN
          DIFF = USCALE(K)
        ELSE
          DIFF = ABS( U(K) - USTART(K) )
        END IF
        USCALE(K) = MAX( DIFF,FTOL )
      END DO
    END IF

!      Initialize to new solution for next loop.

    USTART = U
    FSTART = F
  END IF
! ~~~~~
END DO

NEWTON = NEWTON0
USCALE(:NU) = USCALE0

!-----
!Get derivatives.
!-----

!      Gradients by newton solver if requested.

IF ( (LEVEL0.EQ.1).AND.NEWTON ) THEN
  LEVEL = 1
  ERROR = ERROR0
  USTART = U
  FSTART = F
  CALL OPT_FUNC (LEVEL, U,F,FX,ERROR)

!      Verify that control solution is stable.

```

```

DO K = 1,NU
  DIFF = ABS( U(K) - USTART(K) )
  IF (DIFF.GE.UTOL) THEN
    BADGRAD = .TRUE.
    WRITE (*, '(A91,E10.2,I6,20F8.2)')
+    '(OPT_SOLV) WARNING:  U CHANGED WHEN '//
+    'USING SOLVER TO CALCULATE GRADIENTS FOR DIFF,K,X,W:',
+    DIFF,K,X0(:NX),W0(:NW)
    END IF
  END DO

  DIFF = ABS( F - FSTART )
  IF (DIFF.GE.DIFFMAX) THEN
    BADGRAD = .TRUE.
    WRITE (*, '(A91,E10.2,6X,20F8.2)')
+    '(OPT_SOLV) WARNING:  F CHANGED WHEN '//
+    'USING SOLVER TO CALCULATE GRADIENTS FOR DIFF,X,W: ',
+    DIFF,X0(:NX),W0(:NW)
  END IF

  IF (ERROR.AND.BADGRAD) THEN
    LEVEL = 0
    U = USTART
    F = FSTART
  END IF
ELSE
  LEVEL = 0
END IF

! -----
! Verify outputs.
! -----

NT_OBJ = N_OBJ - NT_OBJ
NT_OPT = N_OPT - NT_OPT
N_SOL = N_SOL + 1
T_SOL = T_SOL + SECNDS(TIME0)

END

```

Subroutine OPT_FUNC

This routine identifies the actual solver used to identify optimal control decisions.

```

SUBROUTINE OPT_FUNC ( LEVEL,
+                   U, F, FX, ERROR )

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Parameters for problem size.
  INTEGER                  LEVEL

  DOUBLE PRECISION          U(NU), F, FX(NX)
  LOGICAL                   ERROR

!-----
!Specifies solver to find optimal controls and resulting cost f(x)
! and (if requested) gradient fx = df/dx.

```

```

!-----
!On input, if ERROR = true on input, diagnostic output is to be provided
! by solver.
!On output, if ERROR = true, there was a problem in solver.
!On input, LEVEL identifies what derivatives are to be calculated
! directly by solver (if it has the capability):
! 0: only u and f
! 1: u, f, and fx
!On output, LEVEL identifies what derivative were actually calculated.
!U on input is used as starting point for solver. This should not
! be changed in this routine since it is already set by the calling
! routines.
!-----

```

```

EXTERNAL          OPT_NPSL, OPT_POLY

```

```

!-----
!
!          -----
!          Verify inputs.
!          -----
!
      IF ( (LEVEL.LT.0).OR.(LEVEL.GT.1) ) THEN
        WRITE (*,*) '(OPT_SOLV) INCONSISTENT LEVEL = ', LEVEL
        STOP '(OPT_FUNC)'
      END IF

!          -----
!          Call solver.
!          -----
!
      IF (NEWTON) THEN
        CALL OPT_NPSL (LEVEL, U,F,FX,ERROR)
      ELSE
        LEVEL = 0
        CALL OPT_POLY (U,F,ERROR)
      END IF

      END

```

Subroutine OPT_NPSL

This routine sets up the data for input into the solver NPSOL.

```

SUBROUTINE OPT_NPSL ( LEVEL,
+                   U, F, FX, ERROR )

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
  INTEGER                   LEVEL

  DOUBLE PRECISION          U(NU), F, FX(NX)
  LOGICAL                   ERROR

!-----
!Solves function value f(x) and, if requested, gradient fx = df/dx for
! node x using NPSOL.

```

```

!On input, LEVEL identifies what derivatives are to be calculated:
! 0: only u and f
! 1: u, f, and fx
!U on input is used as starting point for solver.
!On input, if ERROR = true, printout is provided.
!On output, if ERROR = true, there is a possible problem with
! derivatives and calling routine (solv_opt) will solve gradients by
! finite differences.
!-----
!If LEVEL=0, state variables are treated as constants to avoid extra
! numerical calculation of these gradients while searching for
! optimal control decisions.
!-----

      INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
      INCLUDE 'I.CONTROL'       !Constraints on control.
      INCLUDE 'I.SPECNOW'       !Current stage id.
      INCLUDE 'I.PERFORM'       !Track performance of solver and output.

!-----

!
!           Arrays for transX.

      DOUBLE PRECISION          X(NX), W(NW), S(NW), Y(NX),
+                               YU(NX,NU), YX(NX,NX), YW(NX,NW)

!
!           Arrays for npsol.

      INTEGER                   N, NROWA, NROWJ, NROWR,
+                               INFORM, ITER, ISTATE,
+                               IWORK, LIWORK, LWORK
      DOUBLE PRECISION          OBJF, A, BL, BU, C, CJAC, CLAMDA,
+                               OBJGRAD, R, V, WORK
      ALLOCATABLE                ISTATE(:), IWORK(:), A(:, :), BL(:), BU(:),
+                               C(:), CJAC(:, :), CLAMDA(:),
+                               OBJGRAD(:), R(:, :), V(:), WORK(:)

!
!           Other local variables.

      INTEGER                   LEVEL0, NN, NB, J, NT_OBJ, I, IC, NLCON0
      DOUBLE PRECISION          ADJ(NLCON)
      LOGICAL                   ERROR0, TRIVIAL
      REAL*4                    SECNDS, TIME0

      EXTERNAL                  NPSOL, CONFUN, OBJFUN

!-----

!
!           -----
!           Determine array-size parameters needed by npsol.
!           These will be used in objfun.f.
!           -----

      NN = NU + NX
      NTLIN = NX - NTNLN
      NLCON = NTLIN + NCLIN

      NROWA = MAX (NLCON,1)

```



```

NROWJ = MAX (NCNLN,1)
NROWR = NN
N = NN
LWORK = 2*N**2 + N*NLCON + 2*N*NCNLN + 20*N + 11*NLCON + 21*NCNLN
LIWORK = 3*N + NLCON + 2*NCNLN
NB      = N + NLCON + NCNLN

ALLOCATE ( ISTATE(NB), IWORK(LIWORK), A(NROWA,NN), BL(NB), BU(NB),
+          C(NROWJ), CJAC(NROWJ,NN), CLAMDA(NB),
+          OBJGRAD(NN), R(NROWR,NN), V(NN), WORK(LWORK) )

!          -----
!          Initialize variables.
!          -----

NT_OBJ = N_OBJ
TIME0 = SECNDS(0.0)

LEVEL0 = LEVEL
ERROR0 = ERROR
NLCON0 = NLCON

!          -----
!          Identify linear constraints.
!          -----

!          Add state bounds to linear constraints where
!          transition function is linear.

LEVEL = 1
CALL TRANSXc (LEVEL,NU,NX,NW,NTNLN,
+            ISTAGE,IYEAR,ISEASON,U,X,W,
+            S,Y,YU,YX,YW)
LEVEL = LEVEL0

AA = 0.0
ABL = 0.0
ABU = 0.0

ABL(:NTLIN) = YBL(:NTLIN)
ABU(:NTLIN) = YBU(:NTLIN)
AA(:NTLIN,      :NU)      = YU(:NTLIN,:)
AA(:NTLIN,NU+1  :NU+NX)    = YX(:NTLIN,:)
AA(:NTLIN,NU+NX+1:NU+NX+NW) = YW(:NTLIN,:)

!          Add specified linear constraints.

ABL(NTLIN+1:NLCON) = ACLBL(:NCLIN)
ABU(NTLIN+1:NLCON) = ACLBU(:NCLIN)
AA(NTLIN+1:NLCON, :NU+NX+NW) = ACL(:NCLIN, :NU+NX+NW)

!          -----
!          Identify constraints and initial solution.
!          -----

!          Identify decison bounds.

BL = 0.0

```

```

BU = 0.0
BL(:NU) = UBL(:NU)
BU(:NU) = UBU(:NU)
V(:NU) = U(:NU)

IF (LEVEL.EQ.0) THEN
  N = NU
ELSE
  N = NU + NX
  BL(NU+1:N) = X0(:NX)
  BU(NU+1:N) = X0(:NX)
  V(NU+1:N) = X0(:NX)
END IF

!           Get constraint bound adjustments.

ADJ = 0.0
DO J = 1,NW
  ADJ = ADJ + AA(:NLCON,NU+NX+J)*W0(J)
END DO

IF (LEVEL.EQ.0) THEN
  DO J = 1,NX
    ADJ = ADJ + AA(:NLCON,NU+J)*X0(J)
  END DO
END IF

!           Identify non-trivial constraints.

I = 0
A = 0.0
DO IC = 1,NLCON
  TRIVIAL = .TRUE.
  DO J = 1,N
    IF (AA(IC,J).NE.0.0) TRIVIAL = .FALSE.
  END DO
  IF (.NOT.TRIVIAL) THEN
    I = I + 1
    BL(N+I) = ABL(IC) - ADJ(IC)
    BU(N+I) = ABU(IC) - ADJ(IC)
    A(I,:N) = AA(IC,:N)
  END IF
END DO
NLCON = I

!           -----
!           Call npsol.
!           -----

IWORK = 0
WORK = 0.0
CALL NPSOL ( N, NLCON, NCNLN, NROWA, NROWJ, NROWR,
+          A, BL, BU,
+          CONFUN, OBJFUN,
+          INFORM, ITER, ISTATE,
+          C, CJAC, CLAMDA, OBJF, OBJGRAD, R, V,
+          IWORK, LIWORK, WORK, LWORK )

```



```

        WRITE (*,*) 'ADJUSTED FOR X0 AND W:'
        DO I = 1,NLCON
            WRITE (*, '(2F8.2,A2,30F6.2)') BL(N+I), BU(N+I),
+             ': ', A(I,:N)
        END DO

        ELSE IF (INFORM.EQ.3) THEN
            WRITE (*,*) 'NO FEASIBLE POINT FOUND FOR NONLINEAR
CONSTRAINTS'
        ELSE IF (INFORM.EQ.4) THEN
            WRITE (*,*) 'MAJOR ITERATION LIMIT REACHED'
        ELSE IF (INFORM.EQ.9) THEN
            WRITE (*,*) 'AN INPUT PARAMETER IS INVALID'
        END IF

!           If fatal error, call npsol again to produce
!           diagnostic output before stopping.

        PP = .TRUE.
        if (level.eq.0) then
            write (*, '(a41,10x,i4,f9.4,7x,i5,30f14.8)')
+             '(opt_npsl) inform,time,nOb,u,x,w,f:',
+             inform,secnds(time0),nt_obj,
+             u,x0(:NX),w0(:NW),f
        else
            write (*, '(a44,7x,i4,f9.4,7x,i5,30f14.8)')
+             '(opt_npsl) inform,time,nOb,u,x,w,f,fx:',
+             inform,secnds(time0),nt_obj,
+             u,x0(:NX),w0(:NW),f,fx
        end if
        CALL NPOPTN ('VERIFY LEVEL = 3')
        CALL NPOPTN ('MAJOR PRINT LEVEL = 30')
        CALL NPOPTN ('MINOR PRINT LEVEL = 30')
        CALL NPSOL ( N, NLCON, NCNLN, NROWA, NROWJ, NROWR,
+             A, BL, BU,
+             CONFUN, OBJFUN,
+             INFORM, ITER, ISTATE,
+             C, CJAC, CLAMDA, OBJF, OBJGRAD, R, V,
+             IWORK, LIWORK, WORK, LWORK )

        write(*, '(a16,20i14)') 'istate: ', istate(:n+nlcon)
        write(*, '(a16,20e14.8)') 'objgrad:', objgrad(:n)
        write(*, '(a16,20e14.8)') 'clamda: ', clamda(:n+nlcon)
        write(*, '(a16,i14)') 'inform: ', inform

        STOP '(OPT_NPSL)'
    END IF

!           -----
!           Reset variables.
!           -----

        NLCON = NLCON0

        NT_OBJ = N_OBJ - NT_OBJ
        N_OPT = N_OPT + 1
        T_OPT = T_OPT + SECNDS(TIME0)

```

```

      DEALLOCATE ( ISTATE, IWORK, A, BL, BU,
+               C, CJAC, CLAMDA, OBJGRAD, R, V, WORK )

!               -----
!               Verify outputs.
!               -----

      if (error0.or.error.or.pp.or.p) then
      if (level.eq.0) then
        write (*,'(a41,10x,i4,f9.4,7x,i5,30f14.8)')
+         '(opt_nps1) inform,time,nOb,u,x,w,f:',
+         inform,secnds(time0),nt_obj,
+         u,x0(:NX),w0(:NW),f
      else
        write (*,'(a44,7x,i4,f9.4,7x,i5,30f14.8)')
+         '(opt_nps1) inform,time,nOb,u,x,w,f,fx:',
+         inform,secnds(time0),nt_obj,
+         u,x0(:NX),w0(:NW),f,fx
      end if
      end if

      END

```

Subroutine OPT_POLY

This routine sets up the data for input into the solver AMOEBA.

```

SUBROUTINE OPT_POLY ( U, F, ERROR )

IMPLICIT NONE
INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.

DOUBLE PRECISION          U(NU), F
LOGICAL                   ERROR

!-----
!Solves function value f(x) and using a polytope method,
! more traditionally known as the downhill simplex method
! (not the same as the simplex method of linear optimization).
!-----
!Routine sets up and calls ameoba as solver (from Numerical Recipes).
!U on input is used as starting point for solver.
!If ERROR = true on input, printout is provided.
!Restart is required to confirm all solutions, but should not require
! much computation if already close to solution.
!Does not calculate derivatives.
!-----

      INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
      INCLUDE 'I.CONTROL'      !Constraints on control.
      INCLUDE 'I.SPECNOW'      !Current stage id.
      INCLUDE 'I.PERFORM'      !Track performance of solver and output.

!-----

!               Arrays for transX.

```

```

      INTEGER          LEVEL
      DOUBLE PRECISION X(NX), W(NW), S(NW), Y(NX),
+                     YU(NX,NU), YX(NX,NX), YW(NX,NW)

!               Arrays for ameoba.

      INTEGER          ITER, MP, NP
      DOUBLE PRECISION PM(NU+1,NU), YM(NU+1)

!               Other local variables.

      INTEGER          J, JLO, NT_OBJ
      LOGICAL          ERROR0
      REAL*4           SECNDS, TIME0

      DOUBLE PRECISION OBJVAL
      EXTERNAL         OBJVAL, AMOEBA

!-----

      NT_OBJ = N_OBJ
      TIME0 = SECNDS(0.0)

!-----
!               Add state bounds to linear constraints where
!               transition function is linear.
!-----

      LEVEL = 1
      CALL TRANSXt (LEVEL, NU, NX, NW, NTNLN,
+               ISTAGE, IYEAR, ISEASON, U, X, W,
+               S, Y, YU, YX, YW)

      AA = 0.0
      ABL = 0.0
      ABU = 0.0

      ABL(:NTLIN) = YBL(:NTLIN)
      ABU(:NTLIN) = YBU(:NTLIN)
      AA(:NTLIN, :NU) = YU(:NTLIN, :)
      AA(:NTLIN, NU+1 : NU+NX) = YX(:NTLIN, :)
      AA(:NTLIN, NU+NX+1 : NU+NX+NW) = YW(:NTLIN, :)

!               Add specified linear constraints.

      ABL(NTLIN+1:NLCON) = ACLBL(:NCLIN)
      ABU(NTLIN+1:NLCON) = ACLBU(:NCLIN)
      AA(NTLIN+1:NLCON, :NU+NX+NW) = ACL(:NCLIN, :NU+NX+NW)

!-----
!               Initialize variables.
!-----

      ERROR0 = ERROR
      ERROR = .FALSE.

      MP = NU + 1

```

```

NP = NU

PM(MP,:) = U(:NU)
DO J = 1,NU
  PM(J,:) = PM(MP,:)
  PM(J,J) = PM(MP,J) + USCALE(J)
END DO

DO J = 1,MP
  YM(J) = OBJVAL (PM(J,:))
END DO

! -----
! Call solver.
! -----

CALL AMOEBA (PM,YM,MP,NP,NP,FTOL, OBJVAL,ITER)

! -----
! Save values.
! -----

JLO = 1
DO J = 2,MP
  IF (YM(J).LT.YM(JLO)) JLO = J
END DO

U = PM(JLO,:)
F = YM(JLO)

! -----
! Verify outputs.
! -----

NT_OBJ = N_OBJ - NT_OBJ
N_OPT = N_OPT + 1
T_OPT = T_OPT + SECNDS (TIME0)

END

```

Subroutine OBJFUN

This routine feeds results of OBJ_CALC to the solver NPSOL.

```

SUBROUTINE OBJFUN  ( MODE, N, V, OBJF, OBJGRD, NSTATE )

IMPLICIT NONE
INTEGER          MODE, N, NSTATE
DOUBLE PRECISION OBJF
DOUBLE PRECISION V(N), OBJGRD(N)

! -----
! Provide objective and gradients for NPSOL as a function of
! decision variables u, state variables x, and stochastic variables w.
! -----
! Arrays:  u=v(:NU).  If n=NU, x is fixed and x=x0 passed in common.
! Otherwise, n=NU+NX, x is variables and x=v(NU+1:NU+NX).

```

```
!Gradients: total where  $V(u,x,w) = C(u,x,w) + F(y)$ ,  $y = T(u,x,w)$ ,
! p.d.f. of  $w$  is  $f(x)$ , and  $u$  and  $x$  are independent ( $du/dx = 0$ ).
!-----
```

```
INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
INCLUDE 'I.CONTROL'        !Constraints on control.
```

```
!-----
```

```
INTEGER                LEVEL
DOUBLE PRECISION       U(NU), X(NX), W(NW),
+                      OBJFU(NU), OBJFX(NX)
LOGICAL                P, LARGE

EXTERNAL               OBJ_CALC
```

```
!-----
```

```
!
!-----
! Distinguish between  $x$  fixed and  $x$  variable.
! Also identify what gradients are needed.
!-----
!
```

```
IF (N.EQ.NU) THEN
```

```
  LARGE = .FALSE.
  U = V
  X = X0(:NX)
  W = W0(:NW)
  IF (MODE.EQ.0) THEN
    LEVEL = 0
  ELSE
    LEVEL = 1
  END IF
```

```
ELSE IF (N.EQ.NU+NX) THEN
```

```
  LARGE = .TRUE.
  U = V(1:NU)
  X = V(NU+1:N)
  W = W0(:NW)
  IF (MODE.EQ.0) THEN
    LEVEL = 0
  ELSE
    LEVEL = 1
  END IF
```

```
ELSE
```

```
  WRITE (*,*) '(OBJFUN) N, NU, AND NX DISAGREE:', N, NU, NX
  STOP
```

```
END IF
```

```
!
!-----
! Calculate objective value and gradients.
!-----
!
```



```

CALL OBJ_CALC (LARGE,LEVEL,U,X,W, OBJF,OBJFU,OBJFX )

OBJGRD(:NU) = OBJFU
IF (LARGE) OBJGRD(NU+1:) = OBJFX

END

```

Subroutine OBJVAL

This routine feeds results of OBJ_CALC to the solver AMOEBA.

```

FUNCTION OBJVAL ( U )

IMPLICIT NONE
INCLUDE 'I.SIZEPROB'      !Parameters for problem size.
DOUBLE PRECISION          OBJVAL, U(NU)

!-----
!Returns value of objective function for ameoba, the numerical recipies
! polytope optimization routine.
!-----

INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
INCLUDE 'I.CONTROL'       !Constraints on control.

!-----

INTEGER                  LEVEL
DOUBLE PRECISION          X(NX), W(NW),
+                          OBJF, OBJFU(NU), OBJFX(NX)
LOGICAL                   LARGE

EXTERNAL                  OBJ_CALC

!-----

LARGE = .FALSE.
LEVEL = 0
X = X0(:NX)
W = W0(:NW)
CALL OBJ_CALC (LARGE,LEVEL,U,X,W, OBJF,OBJFU,OBJFX )
OBJVAL = OBJF

END

```

Subroutine OBJ_CALC

This routine solves the objective function as the sum of a current cost and a cost-to-go (evaluated by interpolation). This routine also adds a penalty for constraint violations when using AMOEBA.

```

SUBROUTINE OBJ_CALC ( LARGE, LEVEL, U, X, W,
+                   OBJF, OBJFU, OBJFX )

```

```

IMPLICIT NONE
INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
INTEGER                   LEVEL
DOUBLE PRECISION          U(NU), X(NX), W(NW)
LOGICAL                   LARGE

DOUBLE PRECISION          OBJF, OBJFU(NU), OBJFX(NX)

!-----
!Calculates objective V=objf=C(u,x,w)+F(y) and partial derivatives
! w.r.t. controls u and state x.
!-----
!On input, LEVEL identifies what derivatives are to be calculated:
! 0: only u and f
! 1: u, f, and 1st      (applies only when using NEWTON solver)
!On output, LEVEL identifies if derivatives were calculated
!If LARGE = false, x is fixed and only dV/du calculated.
!If LARGE = true, x is variable and dV/du and dV/dx are calculated.
!-----
!V(u,x,w) = C(u,x,w) + F(y), y = T(u,x,w), p.d.f. of w is f(x).
!Note that u and x are evaluated as independent variables (du/dx=0).
!Note that w and x are evaluated as independent variables (dw/dx=0).
!Also, y=T(u,x,w) is incorporated implicitly:
! dF/du = [dy/du][dF/dy], dF/dx = [dy/dx][dF/dy].
!-----

INCLUDE 'I.SPECNOW'      !Current stage id.
INCLUDE 'I.PERFORM'      !Track performance of solver and output.

!-----

!
!           Local variables for transX.

INTEGER                   VLEVEL
DOUBLE PRECISION          S(NW), Y(NX),
+                           YU(NX,NU), YX(NX,NX), YW(NX,NW)

!
!           Local variables for cost_now.

DOUBLE PRECISION          C, CU(NU), CX(NX), PEN

!
!           Local variables for cost_pen.

DOUBLE PRECISION          CPEN

!
!           Local variables for int_func.

DOUBLE PRECISION          F, FY(NX)

!
!           Other local variables.

INTEGER                   VLEVEL0, J, K
DOUBLE PRECISION          DRMULT, FU(NU), FX(NX)
LOGICAL                   P
REAL                     SECNDS, TIME0, TIME1

EXTERNAL                  TRANSX, INT_FUNC, COST_NOW, COST_PEN

```

```

!-----
!
!           -----
!           Verify inputs.
!           -----
!
IF ( (LEVEL.LT.0).OR.(LEVEL.GT.1) ) THEN
  WRITE (*,*) '(OBJ_CALC) INCONSISTENT LEVEL = ', LEVEL
  STOP
END IF

      N_OBJ = N_OBJ + 1
      TIME0 = SECNDS (0.0)

!
!           -----
!           Identify derivatives needed.
!           -----
!
!           Level identifies if 1st or 2nd derivatives needed.
!           Vlevel identifies if which derivatives are needed,
!           depending on the size of the problem (i.e., if
!           solver is using variable x in order to get df/dx
!           directly from solver vice from finite differences
!           in opt_solv.
!
IF (LEVEL.EQ.0) THEN
  VLEVEL = 0
ELSE
  VLEVEL = 1
  IF (LARGE) VLEVEL = 2
END IF
VLEVEL0 = VLEVEL

!
!           -----
!           Get current cost.
!           -----
!
CALL COST_NOW (VLEVEL,NU,NX,NW,ISTAGE,IYEAR,ISEASON,U,X,W,
+              C,CU,CX,PEN)
IF (VLEVEL.NE.VLEVEL0) THEN
  WRITE (*,*) '(OBJ_CALC) NOT ADAPTED FOR dc BY FINITE DIFF'
  STOP '(OBJ_CALC)'
END IF

!
!           -----
!           Get end-of-stage state y.
!           -----
!
Y = 0.0
YU = 0.0
YX = 0.
CALL TRANSXt (VLEVEL,NU,NX,NW,NTNLN,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             S,Y,YU,YX,YW)
IF (VLEVEL.NE.VLEVEL0) THEN
  WRITE (*,*) '(OBJ_CALC) NOT ADAPTED FOR dy BY FINITE DIFF'
  STOP '(OBJ_CALC)'

```

END IF

!
!
!
!

If using polytope algorithm (i.e., not newton)
then add penalty cost for constraint violation.

IF (.NOT.NEWTON) THEN
CALL COST_PEN (LEVEL,U,X,W,Y,PEN, CPEN)
C = C + CPEN
END IF

!
!
!

Get cost-to-go by interpolation of future cost func.

TIME1 = SECNDS(0.0)

CALL INT_FUNC (LEVEL,Y, F,FY)

T_CALL = T_CALL + SECNDS (TIME1)

!
!
!

Calculate total cost.

DRMULT = 1.0 - DISCOUNT

OBJF = C + F*DRMULT

IF (VLEVEL.GE.1) THEN
FU = 0.0
DO J = 1,NX
FU = FU + YU(J, :)*FY(J)
END DO
OBJFU = CU + FU*DRMULT
END IF

IF (VLEVEL.GE.2) THEN
FX = 0.0
DO J = 1,NX
FX = FX + YX(J, :)*FY(J)
END DO
OBJFX = CX + FX*DRMULT
END IF

!
!
!

Verify outputs.

T_OBJ = T_OBJ + SECNDS (TIME0)

END

Subroutine COST_PEN

This routine evaluates the penalty for constraint violations when using the solver AMOEBA. Because the solver does not include constraints directly, the code uses a large penalty to push control decisions back into the feasible region.

```
SUBROUTINE COST_PEN ( LEVEL, U, X, W, Y, PEN,
+                  CPEN )

  IMPLICIT NONE
  INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
  INTEGER                  LEVEL
  DOUBLE PRECISION         U(NU), X(NX), W(NW), Y(NX), PEN

  DOUBLE PRECISION         CPEN

!-----
!Returns penalty cost of decisions u that violate control bounds or that
! result in a violation state bounds on y given initial state x and
! stage istage.
!-----
!Penalty is a polynomial function of violations
!When applied with polytope solver, penalty consists of linear and
! quadratic terms. Quadratic term assists convergence and linear
! term ensures that constraint is satisfied.
!When applied with Newton-based solver, penalty consists only of
! third order term to ensure continuity of derivatives up to 2nd order.
! Newton-based solvers may not converge without sufficient smoothness.
!-----

  INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
  INCLUDE 'I.CONTROL'       !Constraints on control.
  INCLUDE 'I.PERFORM'       !Track performance of solver and output.

!-----

  INTEGER                  K, J
  DOUBLE PRECISION         DIFF, ADJ(NCLIN)

!-----

!
!          -----
!          Verify inputs consistent with routine.
!          -----
!

  IF ( (.NOT.NEWTN).AND.(LEVEL.NE.0) ) THEN
    WRITE (*,*) '(COST_PEN) LEVEL <> 0 FOR NON-NEWTN SOLVER'
    STOP '(COST_PEN)'
  END IF

  IF (NX-NTNLN+NCLIN.NE.NLCON) THEN
    WRITE (*,*) '(COST_PEN) # LINEAR CONSTRAINTS INCONSISTENT',
+      NX-NTNLN, NCLIN, NLCON
    STOP '(COST_PEN)'
  END IF
```

```

! -----
! Check for violation of bounds and calculate cost.
! -----

CPEN = 0.0

! -----
! Newton solver penalty.
! -----

IF (NEWTON) THEN

  WRITE (*,*) '(COST_PEN) CALLED WITH NEWTON = TRUE'
  STOP '(COST_PEN)'

! -----
! Non-Newton solver penalty.
! -----

ELSE

! Penalty for violating bounds on decisions u.

DO K = 1,NU
  IF (U(K).LT.UBL(K)) THEN
    DIFF = U(K) - UBL(K)
    CPEN = CPEN - PEN*DIFF
  ELSE IF (U(K).GT.UBU(K)) THEN
    DIFF = U(K) - UBU(K)
    CPEN = CPEN + PEN*DIFF
  END IF
END DO

! Penalty for violating bounds on state y.

DO J = 1,NX
  IF (Y(J).LT.YBL(J)) THEN
    DIFF = Y(J) - YBL(J)
    CPEN = CPEN - PEN*DIFF
  ELSE IF (Y(J).GT.YBU(J)) THEN
    DIFF = Y(J) - YBU(J)
    CPEN = CPEN + PEN*DIFF
  END IF
END DO

! Penalty for violating linear constraints.

ADJ = 0.0
DO K = 1,NU
  ADJ = ADJ + ACL(:NCLIN,K)*U(K)
END DO
DO J = 1,NX
  ADJ = ADJ + ACL(:NCLIN,NU+J)*X(J)
END DO
DO J = 1,NW
  ADJ = ADJ + ACL(:NCLIN,NU+NX+J)*W(J)
END DO

```

```

DO K = 1,NCLIN
  IF (ADJ(K).LT.ACLBL(K)) THEN
    DIFF = ADJ(K) - ACLBL(K)
    CPEN = CPEN - PEN*DIFF
  ELSE IF (ADJ(K).GT.ACLBU(K)) THEN
    DIFF = ADJ(K) - ACLBU(K)
    CPEN = CPEN + PEN*DIFF
  END IF
END DO

END IF

END

```

5. MAIN SUBROUTINE AND ACCESSORIES

These routines set up and manage the overall solution of a DDP problem. In addition, the following includes a few accessory routines are used repeatedly for data verification.

Subroutine DYNPROG

This routine control overall flow of the solution, including identification of the model, evaluation of the solution for each stage, and output.

```

SUBROUTINE DYNPROG
  IMPLICIT NONE

  !-----
  !Determines the future-cost (a.k.a. cost-to-go) function for a
  ! stochastic dynamic programming problem.
  !-----
  !Future-cost function is an interpolated function with domain spanned by
  ! nodes stored in a list nnodes long. Each node has the following
  ! associated characteristics:
  !           xn      location of node
  !           fn      function value at node
  !           fxn     function gradient at node
  !           iabove  pointer to nodes above in each dimension
  !           ibelow  pointer to nodes below in each direction
  !Nodes and values are contained in a linked list that points to nodes
  ! immediately above and below in each dimension. The following is a
  ! special case:
  !--Nodes on the edge of the domain:          an adjacent node will be 0
  !-----

  INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
  INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
  INCLUDE 'I.CONTROL'        !Constraints on control.
  INCLUDE 'I.SPECNOW'        !Current stage id.
  INCLUDE 'I.XNODES'         !State discretization.
  INCLUDE 'I.FNODES'         !Future cost function.

```

```

      INCLUDE 'I.SPECW'      !Stochastic realizations.
      INCLUDE 'I.PERFORM'    !Track performance of solver and output.

!-----
!
      Local variables to get cost function in START.DAT.

      INTEGER          IOS, NUR, NXR, ISTAGER, IA, IB
      DOUBLE PRECISION  X, F, FX
      ALLOCATABLE ::     IA(:), IB(:), X(:), FX(:)

!
      Local variables for modelall.

      INTEGER          ILAST
      LOGICAL          RESTART
      CHARACTER*10      LABELS(MAXSEAS), FTOLCHAR

!
      Local variables for modelstg.

      INTEGER          NDX(MAXNX), NNNEW,
+                     IANEW(MAXNX,MAXNODES),
+                     IBNEW(MAXNX,MAXNODES)
      DOUBLE PRECISION XDX(MAXIDX,MAXNX), XNNEW(MAXNX,MAXNODES)
      CHARACTER*11      NAMERUN, NAMECTG

!
      Local variables to hold current cost function.

      DOUBLE PRECISION  FNNEW(MAXNODES), FXNNEW(MAXNX,MAXNODES)

!
      Local variables for node_val.

      INTEGER          LEVEL, IX
      DOUBLE PRECISION U(MAXNU)
      LOGICAL          ERROR

!
      Other local variables.

      INTEGER          LEVEL0, I, J, N
      DOUBLE PRECISION FNMIN, FNAVG, TT_ID, TT_VAL, TT_WEIGH, T_TOT
      REAL             SECNDS, TIME0, TIME1, TIME2
      LOGICAL          P

      EXTERNAL          GETMODEL, NPOPTN, FINALCTG,
+                     NODE_VAL, ADJ_MOD,
+                     EXACT, EXACT_DIFF, OUTPUT

!-----
      TIME0 = SECNDS(0.0)

!-----
!Get parameters that specify characteristics of problem for all stages.
!-----

!
!-----
!
!-----
      Model parameters.
!-----

```



```

!           Parameters are stored in common.

      CALL MODELALL (NU,NX,NW,NTNLN,NCLIN,NCNLN,LPRINT,
+               RESTART,NSTAGES,NSEAS,LABELS,ILAST,IFIRST,IYFIRST,
+               DISCOUNT,TIGHT,FTOL,UTOL,FTOLCHAR,
+               STOCHASTIC,GDP,NEWTON)
      NBASE2 = 2**NX
      NTLIN = NX - NTNLN
      NLCON = NTLIN + NCLIN

!           -----
!           Specify npsol optional parameters.
!           -----

c      CALL NPOPTN ('DIFFERENCE INTERVAL = 0.01')
c      CALL NPOPTN ('DERIVATIVE LEVEL = 3')
c      CALL NPOPTN ('DIFFERENCE INTERVAL = 1.0')
c      CALL NPOPTN ('FUNCTION PRECISION = '//FTOLCHAR)
c      CALL NPOPTN ('CENTRAL DIFFERENCE INTERVAL = 1.0')
      CALL NPOPTN ('VERIFY LEVEL = NO')
c      CALL NPOPTN ('MAJOR ITERATION LIMIT = 100')
      CALL NPOPTN ('MAJOR PRINT LEVEL = 0')
c      CALL NPOPTN ('HESSIAN = YES')

!           -----
!           Identify if gradients needed, and how calculated.
!           -----

!           LEVEL tells if and how gradients are calculated:
!           0: none (only u and f) (not applicable to GDP)
!           1: fx by solver
!           <0: fx by finite diff (even if given by solver)

      IF (GDP) THEN
        LEVEL = 1
      ELSE
        LEVEL = 0
      END IF
      LEVEL0 = LEVEL

!           -----
!           Set up cost function for final stage.
!           -----

!           -----
!           Identify year and season of last stage.
!           -----

      ISTAGE = NSTAGES + 1
      CALL IDNOW (ISTAGE,IFIRST,IYFIRST,NSEAS, IYEAR,ISEASON)

      WRITE (*,*) ('-',I=1,70)
      WRITE (*,*) '(DYNPROG) BEGINNING STAGE ',ISTAGE, ' '//
+               ' SEASON/YEAR: ',ISEASON,'/',IYEAR
      WRITE (*,*) ('-',I=1,70)

!           -----
!           If restart, future cost function from START.DAT.

```

```

! -----
IF (RESTART) THEN

    WRITE (*, '(//,X,43A,A45)') ('-',I=1,40),
+    '(DYNPROG) READING LAST STAGE COST FUNCTION'

!    Open data file with prior solution.

OPEN (10,FILE='START.DAT',STATUS='OLD',IOSTAT=IOS)
IF (IOS.NE.0) THEN
    WRITE (*,*) '(DYNPROG) CANNOT OPEN START.DAT'
    STOP '(DYNPROG)'
END IF
WRITE (*,*) '(DYNPROG) IOSTAT FOR OPENING START.DAT = ', IOS

!    Verify data file consistent with current model.
!    Note: data file can have different discretization.

READ (10, '(4I8,E20.14)') NUR, NXR, NNODES, ISTAGER
IF ( (NUR.NE.NU).OR.(NXR.NE.NX) ) THEN
    WRITE (*,*) '(DYNPROG) INCONSISTENT DATA FILE; NU,NX,NNODES:'
    WRITE (*,*) NUR,NU,NXR,NX
    STOP '(DYNPROG)'
END IF
IF (ISTAGER.NE.NSTAGES) THEN
    WRITE (*,*) '(DYNPROG) LAST STAGE INCONSISTENT WITH DATA FILE'
    WRITE (*,*) ISTAGER, ILAST
    STOP '(DYNPROG)'
C    END IF

!    Read in array values.

ALLOCATE ( IA(NX), IB(NX), X(NX), FX(NX) )
YBL = 1.0E20
YBU = -1.0E20
DO IX = 1,NNODES
    READ (10, '(I8,20E20.14)') J, X
    XN(:NX,IX) = X(:NX)
    DO J = 1,NX
        YBL(J) = MIN (YBL(J),X(J))
        YBU(J) = MAX (YBU(J),X(J))
    END DO
END DO
DO IX = 1,NNODES
    READ (10, '(I8,20I8)') J, IB, IA
    IBELOW(:NX,IX) = IB(:NX)
    IABOVE(:NX,IX) = IA(:NX)
END DO
DO IX = 1,NNODES
    READ (10, '(I8,20E20.14)') J, F, FX
    FN(IX) = F
    FXN(:NX,IX) = FX(:NX)
END DO
DEALLOCATE ( IA, IB, X, FX )

CLOSE (10,IOSTAT=IOS)
WRITE (*,*) '(DYNPROG) IOSTAT FOR CLOSING START.DAT = ', IOS

```

```

! -----
! If not restart, future cost function from finalctg.
! -----

ELSE

      WRITE (*, '(//,X,40A,A48)') ('-',I=1,40),
+      '(DYNPROG) CALCULATING LAST STAGE COST FUNCTION'

!      Get parameters for final cost function.

      CALL MODELSTG (NU,NX,NW,NTNLN,NCLIN,NCNLN,NSTAGES,NSEAS,
+      ISTAGE,IFIRST,IYFIRST,STOCHASTIC,
+      UBL,UBU,UGUESS,USCALE,
+      NDX,NNODES,IBELOW,IABOVE,
+      XBL,XBU,YBL,YBU,XDX,XN,
+      NWNODES,WN,PWN,LIKELY,
+      ACL,ACLBL,ACLB,NAMECTG)

      DO IX = 1,NNODES
        CALL FINALCTG (NX,XN(:NX,IX), FN(IX),FXN(:NX,IX))
          p = .true.
          do j = 1,NX
            if ((xn(j,ix).ne.xbl(j)).and.(xn(j,ix).ne.xbu(j)))
+              p=.false.
          end do
          if (p) write (*, '(a26,i6,20f10.2)')
+            '(dynprog) ix/x/f/fx:',
+            ix,xn(:NX,ix),fn(ix),fxn(:NX,ix)
        END DO

! -----
! Write future cost function for current stage.
! -----

      OPEN (1,IOSTAT=IOS,FILE=NAMERUN//'.dat',STATUS='NEW')
      WRITE (*,*) '(DYNPROG) IOS FOR OPENING '//NAMERUN//'.dat = ', IOS
      WRITE (1,'(6I8)') NU, NX, NNODES, NSTAGES, IYEAR, ISEASON
      DO I = 1,NNODES
        WRITE (1,'(I8,20E20.14)') I, XN(:NX,I)
      END DO
      DO I = 1,NNODES
        WRITE (1,'(I8,20I8)') I, IBELOW(:NX,I), IABOVE(:NX,I)
      END DO
      DO I = 1,NNODES
        WRITE (1,'(I8,20E20.14)') I, FN(I), FXN(:NX,I)
      END DO
      CLOSE (1,IOSTAT=IOS)
      WRITE (*,*) '(DYNPROG) IOS FOR CLOSING '//NAMERUN//'.dat = ', IOS

! -----
! Adjust values as desired.
! -----

      TIME2 = SECNDS(0.0)

      N = NNODES

```

```

        CALL ADJ_MOD (ISTAGE,IYEAR,ISEASON,
+           IBELOW(:NX,:N),IABOVE(:NX,:N),
+           XN(:NX,:N),FN(:N),FXN(:NX,:N))

        WRITE (*,*) 'TIME TO ADJUST STAGE = ', SECNDS(TIME2)

    END IF

!           -----
!           Verify gradients of final_ctg.
!           -----

    IF (NEWTON) THEN

        ! *****

    END IF

!           -----
!           Add nodes as desired for accuracy and convexity.
!           -----

!           Add additional nodes as required to ensure that
!           interpolated values depend only on corner points of
!           surrounding hypercube.

        WRITE (*,*) 'TIME TO SET UP MODEL = ', SECNDS(TIME0)

!-----
!Loop though each stage and state.
!-----

        TT_ID      = 0.0
        TT_WEIGH    = 0.0
        T_TOT       = 0.0

        STAGES: DO ISTAGE = NSTAGES,1,-1
!-----
                TIME1 = SECNDS(0.0)

!           -----
!           Identify year and season of current stage.
!           -----

        CALL IDNOW (ISTAGE,IFIRST,IYFIRST,NSEAS, IYEAR,ISEASON)

        WRITE (*,'(//,70A)') ('-',I=1,70)
        WRITE (*,*) '(DYNPROG) BEGINNING STAGE ', ISTAGE, ' '//
+           ' SEASON/YEAR: ',ISEASON,'/',IYEAR
        WRITE (*,'(70A)') ('-',I=1,70)

!           -----
!           Get parameters for current stage and initialize grid
!           -----

        TIME2 = SECNDS(0.0)

```

```

CALL MODELSTG (NU,NX,NW,NTNLN,NCLIN,NCNLN,NSTAGES,NSEAS,
+           ISTATE,IFIRST,IYFIRST,STOCHASTIC,
+           UBL,UBU,UGUESS,USCALE,
+           NDX,NNNEW,IBNEW,IANEW,
+           XBL,XBU,YBL,YBU,XXNEW,XNNEW,
+           NWNODES,WN,PWN,LIKELY,
+           ACL,ACLBL,ACLB,NAMECTG)

WRITE (*,*) 'TIME TO ID STAGE = ', SECNDS(TIME2)

!
!
!
-----
Get nodes needed for interpolation.
-----

TIME2 = SECNDS(0.0)

WRITE (*,'(X,40A,A45)') ('-',I=1,40),
+       '(DYNPROG) SOLVING GRID VALUES FOR NEW STAGE'

N_INT = 0
N_OBJ = 0
N_OPT = 0
N_SOL = 0
T_ID   = 0.0
T_VAL  = 0.0
T_WEIGH = 0.0
T_INT  = 0.0
T_CALL = 0.0
T_OBJ  = 0.0
T_OPT  = 0.0
T_SOL  = 0.0

!
Start with initial discretization.

DO IX = 1,NNNEW
  LEVEL = LEVEL0
  ERROR = .FALSE.
  U(:NU) = UGUESS(:NU)
  CALL NODE_VALt (LEVEL,IX,XNNEW(:NX,IX),
+               U(:NU),FNNEW(IX),FXNNEW(:NX,IX),ERROR)
END DO

WRITE (*,'(A29,F10.5,I9,F10.2)') 'ID TIME   = ',
+       T_ID/DBLE(N_INT), N_INT, T_ID
WRITE (*,'(A29,F10.5,I9,F10.2)') 'VAL TIME   = ',
+       T_VAL/DBLE(N_INT), N_INT, T_VAL
WRITE (*,'(A29,F10.5,I9,F10.2)') 'WEIGH TIME = ',
+       T_WEIGH/DBLE(N_INT), N_INT, T_WEIGH
WRITE (*,'(A29,F10.5,I9,F10.2)') 'INT TIME   = ',
+       T_INT/DBLE(N_INT), N_INT, T_INT
WRITE (*,'(A29,F10.5,I9,F10.2)') 'CALL INT TIME = ',
+       T_CALL/DBLE(N_INT), N_INT, T_CALL
WRITE (*,'(A29,F10.5,I9,F10.2)') 'OBJ TIME   = ',
+       T_OBJ/DBLE(N_OBJ), N_OBJ, T_OBJ
WRITE (*,'(A29,F10.5,I9,F10.2)') 'OPT TIME   = ',
+       T_OPT/DBLE(N_OPT), N_OPT, T_OPT
WRITE (*,'(A29,F10.5,I9,F10.2)') 'SOLVE TIME = ',
+       T_SOL/DBLE(N_SOL), N_SOL, T_SOL
WRITE (*,'(A29,F10.5,I9,F10.2)') 'NODE TIME   = ',

```

```

+          SECNDS (TIME1) / DBLE (NNODES) , NNODES , SECNDS (TIME1)

      TT_ID    = TT_ID    + T_ID
      TT_VAL   = TT_VAL   + T_VAL
      TT_WEIGH = TT_WEIGH + T_WEIGH
      T_TOT    = T_TOT    + SECNDS (TIME1)

      WRITE (*,*) 'TIME TO SOLVE STAGE = ', SECNDS (TIME2)

!-----
!
! Add nodes as desired for accuracy and convexity.
!-----

! Add additional nodes as required to ensure that
! interpolated values depend only on corner points of
! surrounding hypercube.

!-----

! Update future cost function.
!-----

! Done after developing new function since
! this will replace the old function in common.

NNODES = NNNEW
IABOVE = IANNEW
IBELOW = IBNEW
XN = XNNEW
FN = FNNEW
FXN = FXNNEW

!-----
!
! Adjust values as desired.
!-----

      TIME2 = SECNDS (0.0)

      N = NNODES
      IF (LEVEL0.GT.0) THEN
        CALL ADJ_MOD (ISTAGE,IYEAR,ISEASON,
+          IBELOW(:NX,:N),IABOVE(:NX,:N),
+          XN(:NX,:N),FN(:N),FXN(:NX,:N))
      END IF

      WRITE (*,*) 'TIME TO ADJUST STAGE = ', SECNDS (TIME2)

!-----
!
! Calculate tracking parameters.
!-----

      FNMIN = FN(1)
      FNAVG = FN(1)
      DO IX = 2,NNODES
        FNMIN = MIN (FNMIN,FN(IX))
        FNAVG = FNAVG + FN(IX)
      END DO
      FNAVG = FNAVG/DBLE (NNODES)

```



```

WRITE (*,*) 'TOTAL SOLUTION TIME = ', T_TOT
WRITE (*,*) 'TOTAL TIME = ', SECNDS(TIME0)

!-----
!Final output for first stage (istage = 0).
!-----

      CALL IDNOW (ISTAGE,IFIRST,IYFIRST,NSEAS, IYEAR,ISEASON)

      WRITE (*,*) ('-',I=1,70)
      WRITE (*,*) '(DYNPROG) SUMMARY STAGE ',ISTAGE,' '//
+      '          SEASON/YEAR: ',ISEASON,'/',IYEAR
      WRITE (*,*) ('-',I=1,70)

!-----
!      Get parameters for current stage.
!-----

      CALL MODELSTG (NU,NX,NW,NTNLN,NCLIN,NCNLN,NSTAGES,NSEAS,
+      ISTAGE,IFIRST,IYFIRST,STOCHASTIC,
+      UBL,UBU,UGUESS,USCALE,
+      NDX,NNNEW,IBNEW,IANEW,
+      XBL,XBU,YBL,YBU,DX,XNEW,
+      NWNODES,WN,PWN,LIKELY,
+      ACL,ACLBL,ACLB,NAMECTG)

      NWNODES = 1
      WN(:,1) = 0.0
      PWN(1) = 1.0
      LIKELY = .FALSE.
      LIKELY(1) = .TRUE.

!-----
!      Get solution using median values of stochastic
!      variables.
!-----

      CALL OUTFINAL (NX,NW,ISTAGE,IYEAR,ISEASON, XBL(:NX),XBU(:NX))

      STOP '(DYNPROG) DONE'
      END

```

Subroutine MODELALL

This routine collects the model of a problem and verifies that the data are consistent. Only those data that are independent of the stage are collected by this routine.

```

SUBROUTINE MODELALL ( NU, NX, NW, NTNLN, NCLIN, NCNLN, LPRINT,
+      RESTART, NSTAGES, NSEAS, LABELS,
+      ILAST, IFIRST, IYFIRST,
+      DISCOUNT, TIGHT, FTOL, UTOL, FTOLCHAR,
+      STOCHASTIC, GDP, NEWTON )

      IMPLICIT NONE
      INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
      INTEGER                   NU, NX, NW, NTNLN, NCLIN, NCNLN, LPRINT,
+      NSTAGES, NSEAS, ILAST, IFIRST, IYFIRST

```



```

DOUBLE PRECISION    DISCOUNT, TIGHT, FTOL, UTOL
LOGICAL             RESTART, STOCHASTIC, GDP, NEWTON
CHARACTER*10        LABELS(MAXSEAS), FTOLCHAR

```

```

!-----
!Specify parameters of the system model for all stages.
!-----

```

```

!           Local variables for specprob.

```

```

      INTEGER          NTLIN, NLCON, IYLAST

```

```

!           Other local variables.

```

```

      INTEGER          MMMDV, MMMDIM, MMMWDIM, MMMCON, MMMSEAS, I

```

```

      EXTERNAL         SPECPROB, SIZETEST

```

```

!-----
!           WRITE (*,*) '(MODELALL) BEGIN'

```

```

!           -----
!           Initialize variables.
!           -----

```

```

      MMMDV      = MAXNU
      MMMDIM     = MAXNX
      MMMWDIM    = MAXNW
      MMMCON     = MAXCON
      MMMSEAS    = MAXSEAS

```

```

!           -----
!           Get user specified parameters that describe model.
!           -----

```

```

      CALL SPECPROBt (MMMSEAS,
+                   NU, NX, NW, NTLIN, NTNLN, NCLIN, NCNLN,
+                   RESTART, NSTAGES, NSEAS, LABELS, IFIRST, ILAST,
+                   IYFIRST, IYLAST, LPRINT,
+                   STOCHASTIC, GDP, NEWTON, DISCOUNT, TIGHT, FTOL, UTOL)

```

```

!           Verify parameters specified correctly.

```

```

      CALL SIZETEST (NU,      1, MMMDV,  '(MODELALL) NU      ')
      CALL SIZETEST (NX,      1, MMMDIM, '(MODELALL) NX      ')
      CALL SIZETEST (NW,      0, MMMWDIM, '(MODELALL) NW      ')
      CALL SIZETEST (NTLIN,  0, NX,      '(MODELALL) NTLIN   ')
      CALL SIZETEST (NTNLN,  0, NX,      '(MODELALL) NTNLN   ')
      CALL SIZETEST (NCLIN,  0, MMMCON,  '(MODELALL) MAXCON  ')
      CALL SIZETEST (NCNLN,  0, 0,      '(MODELALL) NCNLN   ')
      CALL SIZETEST (NSTAGES, 0, 10000,  '(MODELALL) NSTAGES ')
      CALL SIZETEST (NSEAS,   1, MMMSEAS, '(MODELALL) NSEAS   ')
      CALL SIZETEST (IFIRST,  1, NSEAS,  '(MODELALL) IFIRST  ')
      CALL SIZETEST (ILAST,   1, NSEAS,  '(MODELALL) ILAST   ')

```

```

      IF (NTLIN+NTNLN.NE.NX) THEN
        WRITE (*,*) '(MODELALL) NTLIN + NTNLN <> NX'

```

```

      STOP '(MODELALL)'
END IF

NLCON = NTLIN + NCLIN
IF (NLCON.GT.MMMCON) THEN
  WRITE (*,*) '(MODELALL) NLCON > MAXCON'
  STOP '(MODELALL)'
END IF

IF (.NOT.RESTART) THEN
  IF (NSTAGES.EQ.0) THEN
    WRITE (*,*) '(MODELALL) NSTAGES = 0 WITHOUT RESTART'
    STOP '(MODELALL)'
  END IF

  I = IYFIRST + INT ((IFIRST+NSTAGES-1)/NSEAS)
  IF (IYLAST.NE.I) THEN
    WRITE (*,*) '(MODELALL) IYFIRST AND IYLAST DISAGREE',
+     IYFIRST, IYLAST, I
    STOP '(MODELALL)'
  END IF

  I = MOD (IFIRST+NSTAGES-1,NSEAS) + 1
  IF (I.EQ.NSTAGES+1) I = 1
  IF (I.NE.ILAST) THEN
    WRITE (*,*) '(MODELALL) IFIRST AND ILAST DISAGREE',
+     IFIRST, ILAST, I
    STOP '(MODELALL)'
  END IF
END IF

IF ( STOCHASTIC.AND.(NW.LE.0) ) THEN
  WRITE (*,*) '(MODELSTG) STOCHASTIC MODEL WITH NW <= 0'
  STOP '(MODELALL)'
END IF

IF ( (.NOT.GDP).AND.NEWTON ) THEN
  WRITE (*,*) '(MODELALL) MULTILINEAR DP CANNOT USE NEWTON SOLVER'
  STOP '(MODELALL)'
END IF

IF ( (DISCOUNT.LT.0.0).OR.(DISCOUNT.GT.1.0) ) THEN
  WRITE (*,*) '(MODELALL) DISCOUNT NOT IN [0,1],', DISCOUNT
  STOP '(MODELALL)'
END IF

IF (TIGHT.LT.0.0) THEN
  WRITE (*,*) '(MODELALL) TIGHT < 0,', TIGHT
  STOP '(MODELALL)'
END IF

WRITE (*,*) 'SPECPROB OK'

!
! -----
! Get character string for ftol.
! -----

OPEN (1,STATUS='SCRATCH')

```

```

WRITE (1, '(E10.2)') FTOL
REWIND (1)
READ (1, '(A10)') FTOLCHAR
CLOSE (1)

```

```

!-----
!Echo model parameters.
!-----

```

```

WRITE (*,*) ('-',I=1,40)
WRITE (*,*) '(MODELALL) USER SPECIFIED MODEL'
WRITE (*,*) ('-',I=1,40)
WRITE (*,*)

WRITE (*,*) 'DISCOUNT RATE =', DISCOUNT
WRITE (*,*)

WRITE (*,*) 'GDP, TIGHT =', GDP, TIGHT
WRITE (*,*) 'NEWTON SOLV=', NEWTON
WRITE (*,*) 'STOCHASTIC =', STOCHASTIC
WRITE (*,*) 'PRECISION OF OBJECTIVE FUNCTION =', FTOL
WRITE (*,*) 'PRECISION OF CONTROLS =', UTOL
WRITE (*,*)

WRITE (*,*) 'RESTART = ', RESTART
WRITE (*,*) '#STAGES=', NSTAGES, ', #SEASONS =', NSEAS
WRITE (*,*) 'DATES (SEASON/YEAR): ',
+         IFIRST, '/', IYFIRST, '--',
+         ILAST, '/', IYLAST
WRITE (*,*) 'YEARS OF OPERATION =',
+         DBLE(NSTAGES)/DBLE(NSEAS)
WRITE (*,*)

WRITE (*,*) '#DECISION VARIABLES =', NU
WRITE (*,*) '#STATE VARIABLES =', NX
WRITE (*,*) '#STOCHASTIC VARIABLES =', NW
WRITE (*,*) '#LINEAR CONSTRAINTS =', NCLIN
WRITE (*,*)

WRITE (*,*) 'LPRINT =', LPRINT
WRITE (*,*)

WRITE (*,*) '(MODELALL) END'

```

END

Subroutine MODELSTG

This routine collects the model of a problem and verifies that the data are consistent. Those data that can change with the stage are collected by this routine. This routine is structured to allow adaptive grids in later work.

```

SUBROUTINE MODELSTG ( NU, NX, NW, NTNLN, NCLIN, NCNLN, NSTAGES,
+                     NSEAS, ISTAGE, IFIRST, IYFIRST, STOCHASTIC,
+                     UBL, UBU, UGUESS, USCALE,
+                     NDX, NNODES, IBELOW, IABOVE,

```

```

+           XBL, XBU, YBL, YBU, XDX, XN,
+           NWNODES, WN, PWN, LIKELY,
+           ACL, ACLBL, ACLBU, NAMERUN, NAMECTG )

IMPLICIT NONE
INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
INTEGER                   NU, NX, NW, NTNLN, NCLIN, NCNLN,
+                           NSTAGES, NSEAS, ISTAGE, IFIRST, IYFIRST
LOGICAL                   STOCHASTIC

INTEGER                   NDX(MAXNX), NNODES,
+                           IBELOW(MAXNX,MAXNODES),
+                           IABOVE(MAXNX,MAXNODES),
+                           NWNODES
DOUBLE PRECISION          UBL(MAXNU), UBU(MAXNU),
+                           UGUESS(MAXNU), USCALE(MAXNU),
+                           XBL(MAXNX), XBU(MAXNX),
+                           YBL(MAXNX), YBU(MAXNX),
+                           XDX(MAXIDX,MAXNX),
+                           XN(MAXNX,MAXNODES),
+                           WN(MAXNW,MAXWNODES), PWN(MAXWNODES),
+                           ACL(MAXCON,MAXNU+MAXNX+MAXNW),
+                           ACLBL(MAXCON), ACLBU(MAXCON)
LOGICAL                   LIKELY(MAXWNODES)
CHARACTER*11              NAMERUN, NAMECTG

```

```

!-----
!Specify parameters of the system model for the current stage.
!-----
!Parameters of the stochastic model (e.g, for streamflow) are applied to
! the multivariate random normal variables in the w transition function.
!-----

```

```

INCLUDE 'I.PERFORM'      !Track performance of solver and output.

```

```

!-----
!           Local variables for specU.

```

```

INTEGER                   IYEAR, ISEASON

```

```

!           Local variables for specX.

```

```

INTEGER                   MMMIDX, KSTAGE, KYEAR, KSEASON

```

```

!           Local variables for specW.

```

```

INTEGER                   MMMIDW, MODELW(NW), NDW(NW)
DOUBLE PRECISION          WMEAN(NW), WSTDV(NW), WSKEW(NW),
+                           WDW(MAXIDW,NW), PROBW(MAXIDW,NW),
+                           SWLO(NW), SWHI(NW), PROBMIN
LOGICAL                   GAUSQUAD

```

```

!           Other local variables.

```

```

INTEGER                   J, IDX, ID(MAXNX), IBASENX(MAXNX),
+                           I, IW, IXB, N,
+                           N1, N2, N3, N4, N5, N6, N7

```

```

DOUBLE PRECISION      DX(MAXIDX,MAXNX), ALARGE, VALOLD, PTEST
LOGICAL               P

EXTERNAL              IDNOW, SPECU, SPECX, SPECW, SPECLCON,
+                     SIZETEST, SW_GQ, SW_TRAP

!-----

      WRITE (*,*) '(MODELSTG) BEGIN'

      P = .FALSE.
      IF (LPRINT.GE.1) P = .TRUE.
      IF (ISTAGE.EQ.NSTAGES+1) P = .TRUE.

!-----
!Get user specified parameters that describe model.
!-----

      ALARGE = 1.0E+20

!-----
!                               ID year and season of current and following stage.
!-----

      CALL IDNOW (ISTAGE,IFIRST,IYFIRST,NSEAS, IYEAR,ISEASON)
      KSTAGE = ISTAGE + 1
      CALL IDNOW (KSTAGE,IFIRST,IYFIRST,NSEAS, KYEAR,KSEASON)

!-----
!                               Get number of decision variables, bounds, and guess.
!-----

      UBL = -ALARGE
      UBU = ALARGE
      N = NU
      CALL SPECU (NU,ISTAGE,IYEAR,ISEASON,
+               UBL(:N),UBU(:N),UGUESS(:N),USCALE(:N))

!-----
!                               Verify parameters specified correctly.
!-----

      DO J = 1,NU
        IF (UBL(J).GE.UBU(J)) THEN
          WRITE (*,*) '(MODELSTG) UB: ', UBL(J), ' NOT < ', UBU(J)
          STOP '(MODELSTG)'
        END IF
      END DO

      IF (P) WRITE (*,*) 'SPECU OK'

!-----
!                               Get bounds on end-of-stage state variables.
!-----

      MMMIDX = MAXIDX
      N = NX
      CALL SPECX (MMMDX,NX,KSTAGE,KYEAR,KSEASON,
+               YBL(:N),YBU(:N),NDX(:N),XDX(:, :N))

```

```

!                               Verify parameters specified correctly.

DO J = 1,NX
  IF ((YBL(J).NE.XDX(1,J)).OR.(YBU(J).NE.XDX(NDX(J),J))) THEN
    WRITE (*,*) '(MODELSTG) Y BOUNDS MUST BE AT HI AND LOW NODES'
    WRITE (*,'(11X,4A10)') 'DIM', 'NX','BOUND','NODE'
    WRITE (*,'(11X,2I10,2F10.2)') J, 1, YBL(J), XDX(1,J)
    WRITE (*,'(11X,2I10,2F10.2)') J, NDX(J), YBU(J), XDX(NDX(J),J)
    STOP '(MODELSTG)'
  END IF
END DO

!                               -----
!                               Get bounds on state variables and (initial)
!                               discretization.
!                               -----

MMMIDX = MAXIDX
N = NX
CALL SPECX (MMMIDX,NX,ISTAGE,IYEAR,ISEASON,
+          XBL(:N),XBU(:N),NDX(:N),XDX(:, :N))

!                               Verify parameters specified correctly.

DO J = 1,NX
  IF ((XBL(J).NE.XDX(1,J)).OR.(XBU(J).NE.XDX(NDX(J),J))) THEN
    WRITE (*,*) '(MODELSTG) X BOUNDS MUST BE AT HI AND LOW NODES'
    WRITE (*,'(11X,A10,A10)') 'BOUND', 'NODE'
    WRITE (*,'(11X,F10.2,F10.2)') XBL(J), XDX(1,J)
    WRITE (*,'(11X,F10.2,F10.2)') XBU(J), XDX(NDX(J),J)
    STOP '(MODELSTG)'
  END IF
END DO

DX = 0.0
DO J = 1,NX
  IF (NDX(J).LT.2) THEN
    WRITE (*,*) NDX(:N)
    WRITE (*,*) '(MODELSTG) NDX NOT > 2'
    STOP '(MODELSTG)'
  END IF
  DO IDX = 1,NDX(J)-1
    DX(IDX,J) = XDX(IDX+1,J) - XDX(IDX,J)
    IF (DX(IDX,J).LE.0.0) THEN
      WRITE (*,'(20E8.2)') XDX(IDX:IDX+1,J), DX(IDX,J)
      WRITE (*,*) '(MODELSTG) DX < 0: XDX OR NDX INCORRECT',J,IDX
      STOP '(MODELSTG)'
    END IF
  END DO
END DO

NNODES = 1
DO J = 1,NX
  NNODES = NNODES * NDX(J)
END DO
N = MAXNODES
CALL SIZETEST (NNODES,1,N, '(MODELSTG) NNODES ')
DO J = 1,NX

```

```

      CALL SIZETEST (NDX(J),1,MMMIDW,      '(MODELSTG) IDX      ')
END DO

IF (P) WRITE (*,*) 'SPECX OK'

!           -----
!           Get stochastic variables, discretization, and
!           significance level required.
!           -----

MODELW = 0
WMEAN = ALARGE
WSTDV = ALARGE
WSKEW = ALARGE
NDW = 0
WDW = ALARGE
PROBW = ALARGE
PROBMIN = 0.0
MMMIDW = MAXIDW
N = NW
CALL SPECWt (MMMIDW,NW,ISTAGE,IYEAR,ISEASON,
+           MODELW,WMEAN,WSTDV,WSKEW,
+           NDW,WDW,PROBW,SWLO,SWHI,PROBMIN,GAUSQUAD)

!           Verify parameters specified correctly.

DO J = 1,NW
  IF (WMEAN(J).EQ.ALARGE) THEN
    WRITE (*,*) '(MODELSTG) WMEAN NOT ASSIGNED FOR W('',J,'')'
    STOP '(MODELSTG)'
  END IF
  IF (STOCHASTIC) THEN
    CALL SIZETEST (MODELW(J),1,2,      '(MODELSTG) MODELW      ')
    CALL SIZETEST (WSTDV(J),0,ALARGE, '(MODELSTG) WSTDV      ')
    IF (WSTDV(J).EQ.ALARGE) THEN
      WRITE (*,*) '(MODELSTG) WSTDV NOT ASSIGNED FOR W('',J,'')'
      STOP '(MODELSTG)'
    END IF
    IF (MODELW(J).GE.3) THEN
      IF (WSKEW(J).EQ.ALARGE) THEN
        WRITE (*,*) '(MODELSTG) WSKEW NOT ASSIGNED FOR W('',J,'')'
        STOP '(MODELSTG)'
      END IF
    END IF
  END IF

  VALOLD = -ALARGE
  PTEST = 0.0
  N = MAXIDW
  CALL SIZETEST (NDW(J),1,MMMIDW,      '(MODELSTG) NDW      ')
  DO I = 1,NDW(J)
    IF (WDW(I,J).EQ.ALARGE) THEN
      WRITE (*,*) '(MODELSTG) WDW NOT ASSIGNED, W('',I,J,'')'
      STOP '(MODELSTG)'
    END IF
    IF (WDW(I,J).LE.VALOLD) THEN
      WRITE (*,*) '(MODELSTG) WDW NOT IN ORDER, W('',I,J,'')'
      STOP '(MODELSTG)'
    END IF
  END DO
END DO

```

```

        VALOLD = WDW(I,J)
        IF (PROBW(I,J).EQ.ALARGE) THEN
            WRITE (*,*) '(MODELSTG) PROBW NOT ASSIGNED, W('',I,J,'')'
            STOP '(MODELSTG)'
        END IF
        PTEST = PTEST + PROBW(I,J)
    END DO
    IF (ABS(PTEST-1.0).GT.PROBMIN) THEN
        WRITE (*,*) '(MODELSTG) PTEST <> 1.0,', J, PTEST-1.0
        STOP '(MODELSTG)'
    END IF

    IF (.NOT.GAUSQUAD) THEN
        IF (SWLO(J).GE.SWHI(J)) THEN
            WRITE (*,*) '(MODELSTG) SWLO > SWHI,', J
            STOP '(MODELSTG)'
        END IF
    END IF
END IF
END IF
END IF
END DO

IF (STOCHASTIC) THEN
    N = NINT(PROBMIN)
    CALL SIZETEST (N,0,1,                      '(MODELSTG) PROBMIN ')
    NWNODES = 1
    DO J = 1,NW
        NWNODES = NWNODES * NDW(J)
    END DO
    N = MAXNWNODES
    CALL SIZETEST (NWNODES,2,N,                '(MODELSTG) NWNODES ')
ELSE
    NDW = 1
    NWNODES = 1
END IF

IF (P) WRITE (*,*) 'SPECW OK'

! -----
! Get linear constraints on end-of-stage state.
! -----

ACLBL = -ALARGE
ACLBU = ALARGE
CALL SPECLCON (NU,NX,NW,NCLIN,KSTAGE,KYEAR,KSEASON,
+             ACL(:NCLIN,:NU+NX+NW),
+             ACLBL(:NCLIN),ACLBU(:NCLIN))

! Verify parameters specified correctly.

DO I = 1,NCLIN
    IF (ACLBL(I).GE.ACIBU(I)) THEN
        WRITE (*,*) '(MODELSTG) ERROR BOUNDS: ACLBL NOT < ACIBU'
        WRITE (*,*) 'ENDING STAGE =', KSTAGE,KYEAR,KSEASON
        WRITE (*,*) ACLBL(I), ' NOT < ', ACIBU(I)
        STOP '(MODELSTG)'
    END IF
END DO

```



```

! -----
! Get linear constraints.
! -----

ACLBL = -ALARGE
ACLBU = ALARGE
CALL SPECLCON (NU,NX,NW,NCLIN,ISTAGE,IYEAR,ISEASON,
+             ACL(:NCLIN,:NU+NX+NW),
+             ACLBL(:NCLIN),ACLBU(:NCLIN))

! Verify parameters specified correctly.

DO I = 1,NCLIN
  IF (ACLBL(I).GE.ACBLU(I)) THEN
    WRITE (*,*) '(MODELSTG) ERROR BOUNDS: ACLBL NOT < ACLBU'
    WRITE (*,*) '          STAGE =', ISTAGE,IYEAR,ISEASON
    WRITE (*,*) ACLBL(I), ' NOT < ', ACLBU(I)
    STOP '(MODELSTG)'
  END IF
END DO

WRITE (*,*) 'SPECLCON OK'

! -----
! Convert values to variables needed and evaluate other values that need
! to be calculated only once.
! -----

! -----
! Get character string to identify current run.
! -----

IF ( (NDX(1).GT.99).OR.(NDW(1).GT.9).OR.(ISTAGE+1.GT.999) ) THEN
  WRITE (*,*) '(MODELSTG) ERROR IN CREATING FILE NAMES'
  WRITE (*,*) NDX(1), NDW(1), ISTAGE
  STOP '(MODELSTG)'
END IF
IF ( (NX.GT.9).OR.(NW.GE.9) ) THEN
  WRITE (*,*) '(MODELSTG) ERROR IN CREATING FILE NAMES', NX, NW
  STOP '(MODELSTG)'
END IF

N1 = INT(NDX(1)/10)
N2 = NDX(1) - 10*N1
N3 = INT(NDW(1)/10)
N4 = NDW(1) - 10*N3

OPEN (1,STATUS='SCRATCH')

N5 = INT( ISTAGE/100 )
N6 = INT( (ISTAGE - 100*N5)/10 )
N7 = INT( ISTAGE - 100*N5 - 10*N6 )
REWIND (1)
WRITE (1,'(A1,3I1,A1,3I1,A1,2I1)')
+ 't',N5,N6,N7,'x',NX,N1,N2,'w',NW,N4
REWIND (1)
READ (1,'(A11)') NAMERUN

```

```

N5 = INT( (ISTAGE + 1)/100 )
N6 = INT( (ISTAGE + 1 - 100*N5)/10 )
N7 = INT( ISTAGE + 1 - 100*N5 - 10*N6 )
REWIND (1)
WRITE (1, '(A1,3I1,A1,3I1,A1,2I1)')
+      't',N5,N6,N7,'x',NX,N1,N2,'w',NW,N4
REWIND (1)
READ (1, '(A11)') NAMECTG

CLOSE (1)

!      -----
!      Get state-space grid.
!      -----

IBASENX(1) = 1
DO J = 2,NX
  IBASENX(J) = IBASENX(J-1)*NDX(J-1)
END DO

!      No links initially:  added as they are created.

IBELOW = -1
IABOVE = -1

!      Positions and pointers to adjacent nodes.
!      On the edge of the domain, adjacent node will be 0.

IBELOW(:NX,:NNODES) = 0
IABOVE(:NX,:NNODES) = 0

ID = 1
ID(1) = 0
DO I = 1,NNODES

  DO J = 1,NX
    ID(J) = ID(J) + 1
    IF (ID(J).GT.NDX(J)) THEN
      ID(J) = 1
    ELSE
      EXIT
    END IF
  END DO

  DO J = 1,NX
    XN(J,I) = XDX(ID(J),J)
    IF (ID(J).GT.1) THEN
      IXB = I - IBASENX(J)
      IBELOW(J,I) = IXB
      IABOVE(J,IXB) = I
    END IF
  END DO

END DO

!      -----
!      Calculate locations and weights for each dimension.
!      -----

```

```

      IF (STOCHASTIC) THEN
!
!           #####
!
      ELSE
        NDW = 1
        WDW(1,:NW) = WMEAN(:NW)
        PROBW = 0.0
        PROBW(1,:NW) = 1.0
      END IF

!
!           -----
!           Get multivariate locations and weights for each node
!           -----
!
      PWN = 0.0
      LIKELY = .FALSE.
      IF (GAUSQUAD) LIKELY = .TRUE.
      ID = 1
      ID(1) = 0
      DO IW = 1,NWNODES

!           Identify current node's position.

        DO J = 1,NW
          ID(J) = ID(J) + 1
          IF (ID(J).GT.NDW(J)) THEN
            ID(J) = 1
          ELSE
            EXIT
          END IF
        END DO

!           Identify and calculate node's probability weight.

        PTEST = 1.0
        DO J = 1,NW
          WN(J,IW) = WDW(ID(J),J)
          PTEST = PTEST*PROBW(ID(J),J)
        END DO
        PWN(IW) = PTEST

!           Identify if node is likely.

        IF (PTEST.GT.PROBMIN) LIKELY(IW) = .TRUE.

      END DO

!           Verify probabilities sum to one.

      PTEST = 0.0
      DO IW = 1,NWNODES
        PTEST = PTEST + PWN(IW)
      END DO
      IF (ABS(1.0-PTEST).GT.PROBMIN) THEN
        WRITE (*,*) '(MODELSTG) PROB < 1.0', PTEST, 1.0-PTEST
        STOP '(MODELSTG)'
      END IF

```

END IF

! Verify likely probabilities sum to one.

```
PTEST = 0.0
DO IW = 1,NWNODES
  IF (LIKELY(IW)) PTEST = PTEST + PWN(IW)
END DO
IF (ABS(1.0-PTEST).GT.10.0*PROBMIN) THEN
  WRITE (*,*) '(MODELSTG) LIKELY PROB < 1.0', PTEST, 1.0-PTEST
  STOP '(MODELSTG)'
END IF
```

!-----
!Echo model parameters.
!-----

```
IF (P) THEN
  WRITE (*,*) '(X,20A,A11,20A,A40)')
  ('-',I=1,20), NAMERUN, ('-',I=1,20),
  '(MODELSTG) MODEL FOR STAGE'

  WRITE (*,*) '#DECISION VARIABLES =', NU
  WRITE (*,*) '(A8,20F8.2)') 'MIN U:', UBL(:NU)
  WRITE (*,*) '(A8,20F8.2)') 'MAX U:', UBU(:NU)
  WRITE (*,*) '(A12,20F8.2)') 'TRIAL SOLN:', UGUESS(:NU)
  WRITE (*,*) '(A14,20F8.2)') 'LENGTH SCALE:', USCALE(:NU)
  WRITE (*,*)

  WRITE (*,*) '#STATE VARIABLES =', NX
  WRITE (*,*) '(A8,20F8.2)') 'MIN X:', XBL(:NX)
  WRITE (*,*) '(A8,20F8.2)') 'MAX X:', XBU(:NX)
  WRITE (*,*) '(A16,20I4)') 'DISCRETIZATION:', NDX(:NX)
  WRITE (*,*) 'TOTAL INITIAL NODES =', NNODES
  DO J = 1,NX
    WRITE (*,*) '(A6,I2,A2,20F8.2)')
    'XDX(',J,'):',XDX(:NDX(J),J)
  END DO

  DO J = 1,NX
    WRITE (*,*) '(A6,I2,A2,20F8.2)')
    'DX(',J,'):',DX(:NDX(J)-1,J)
  END DO
  WRITE (*,*)

  WRITE (*,*) '#STOCHASTIC VARIABLES =', NW
  WRITE (*,*) '(A16,20I4)') 'DISCRETIZATION:', NDW(:NW)
  WRITE (*,*) 'TOTAL NODES =', NWNODES
  WRITE (*,*) 'GAUSSIAN QUADRATURE =', GAUSQUAD
  IF (GAUSQUAD) THEN
    WRITE (*,*) '(A12,A10,A18)')
    'IW','PROB','DISCRETE STDV''S'
    DO I = 1,NWNODES
      WRITE (*,*) '(6X,I6,F10.6,20F6.2)')
      I, PWN(I), WN(:NW,I)
    END DO
  ELSE
    WRITE (*,*) '(A12,A8,A8,A18)')
```

```

+           'IW', 'PROB', 'LIKELY?', 'DISCRETE STDV' 'S'
      DO I = 1, NWNODES
        WRITE (*, '(6X, I6, F8.5, L8, 20F6.2)')
+           I, PWN(I), LIKELY(I), WN(:NW, I)
      END DO
      END IF
      WRITE (*, *)

      WRITE (*, *) '#LINEAR CONSTRAINTS =', NCLIN
      IF (NCLIN.NE.0) WRITE (*, *) 'BL, BU :: A'
      DO I = 1, NCLIN
        WRITE (*, '(2F8.2, A2, 30F6.2)') ACLBL(I), ACLBU(I),
+           ' :: ', ACL(I, :NU+NX+NW)
      END DO
      WRITE (*, *)

      WRITE (*, *) '(MODELSTG) END'
      END IF

      END

```

Subroutine NODE_VAL

This routine identifies the expected cost-to-go and first derivatives for an initial state. This includes evaluation of control decisions and costs for each outcome of the stochastic variables (i.e., nodes of the stochastic-space grid). Gradients are evaluated by a crude finite-difference method if needed and not provided by other routines.

```

      SUBROUTINE NODE_VALt ( LEVEL, IXNODE, X,
+                           U, F, FX, ERROR )

      IMPLICIT NONE
      INCLUDE 'I.SIZEPROB'      !Problem size parameters and tolerances.
      INTEGER                   LEVEL, IXNODE
      DOUBLE PRECISION          X(NX)

      DOUBLE PRECISION          U(NU), F, FX(NX)
      LOGICAL                   ERROR

      !-----
      !Calls user specified solver to find optimal control u and associated
      ! cost-function value f(x) for initial state x. If requested, gradient
      ! fx = df/dx is calculated.
      !-----
      !Values calculated as weighted average of stochastic variable outcomes.
      !Gradients taken directly from solver if they appear correct; otherwise,
      ! gradient calculated by finite differences.
      !On input, LEVEL identifies what derivatives are to be calculated:
      ! 0: none (only u and f)
      ! 1: fx
      !On output, LEVEL identifies how derivatives actually were calculated
      ! (if derivatives requested).
      !On input, if ERROR = true on input, diagnostic output is to be provided
      ! by solver.
      !On output, if ERROR = true, there was a problem with solver.

```

```

!U on input identifies starting solution.  Not reinitialized for current
! x on guess that solutions for different iw will be closer than uguess.
!U on output identifies weighted average of solutions for each iw.  It
! DOES NOT identify an actual solution unless deterministic (nwnodes=1).
!-----

```

```

      INCLUDE 'I.SIZEALLO'      !Parameters to allocate storage space.
      INCLUDE 'I.CONTROL'       !Constraints on control.
      INCLUDE 'I.SPECW'        !Stochastic realizations.
      INCLUDE 'I.SPECNOW'      !Current stage id.
      INCLUDE 'I.PERFORM'      !Track performance of solver and output.

```

```

!-----

```

```

!               Local variables for transx.

```

```

      DOUBLE PRECISION      S(NW), Y(NX),
+                          YU(NX,NU), YX(NX,NX), YW(NX,NW)

```

```

!               Local variables cost_now.

```

```

      DOUBLE PRECISION      C, CU(NU), CX(NX), PEN

```

```

!               Local variables for transw.

```

```

      DOUBLE PRECISION      SW(NW), W(NW)

```

```

!               Local variables for opt_solv.

```

```

      LOGICAL               ERRTEST

```

```

!               Local variables for finite difference calculations.

```

```

      INTEGER               J, L
      DOUBLE PRECISION      UHOLD(NU), FHOLD,
+                          DEL, DELINV1, DELINV2, DELINV,
+                          XDEL(NX), F1, F2, U1(NU), U2(NU)
      LOGICAL               SKIP

```

```

!               Other local variables.

```

```

      INTEGER               IW, LEVEL0, K,
+                          NT_SOL, NT_OPT, NT_OBJ, NT_INT
      DOUBLE PRECISION      XHOLD(NX), WHOLD(NW),
+                          DUMMY, SUMF, SUMFX(NX), SUMU(NU),
+                          ADJ(NLCON),
+                          ERRTOL, ERRTST, G,
+                          C1, C2, Y1(NX), Y2(NX)
      LOGICAL               FDIFF, ERROR0
      REAL*4                SECNDS, TIME0

      EXTERNAL              COST_NOW, TRANSX, OPT_SOLV

```

```

!-----

```

```

!               Specify interval for finite differences.
!               Used for override of solver gradients and for
!               verification of gradients in cost_now and transx.

```

```

DEL      = 1.0E-08

!              Specify tolerance for finite diff verification.

ERRTOL = 1.0E-04

!              -----
!              Verify inputs.
!              -----

IF ( (LEVEL.LT.0).OR.(LEVEL.GT.1) ) THEN
  WRITE (*,*) '(NODE_VAL) INCONSISTENT LEVEL = ', LEVEL
  STOP
END IF

IF (NTNLN.NE.0) THEN
  WRITE (*,*) '(NODE_VAL) NOT ADAPTED FOR NON-LINEAR TRANSITION'
  STOP '(NODE_VAL)'
END IF

!              -----
!              Save initial settings.
!              -----

      NT_INT = N_INT
      NT_OBJ = N_OBJ
      NT_OPT = N_OPT
      NT_SOL = N_SOL

ERROR0 = ERROR
LEVEL0 = LEVEL
XHOLD = X

!              -----
!              Initialize values.
!              -----

      TIME0 = SECNDS(0.0)

DUMMY = 9.9E20
DELINV1 = 1.0 / DEL
DELINV2 = 0.5 * DELINV1

!-----
!Find optimal u and cost for each w.
!-----

SUMU  = 0.0
SUMF  = 0.0
SUMFX = 0.0
ERROR = .FALSE.

DO IW = 1,NWNODES
  IF (LIKELY(IW)) THEN
    WHOLD = WN(:NW,IW)
    W = WHOLD

```

!
!
!

Check current cost function.

```
LEVEL = 1
C = DUMMY
CU = DUMMY
CX = DUMMY
PEN = DUMMY
CALL COST_NOW (LEVEL,NU,NX,NW,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             C,CU,CX,PEN)
IF (LEVEL.NE.1) THEN
  WRITE (*,*) '(NODE_VAL) NOT ADAPTED FOR dc BY FINITE DIFF'
  STOP '(NODE_VAL)'
END IF
IF (C.EQ.DUMMY) THEN
  WRITE (*,*) '(NODE_VAL) C MISSING FROM COST_NOW'
  STOP '(NODE_VAL)'
END IF
IF (PEN.EQ.DUMMY) THEN
  WRITE (*,*) '(NODE_VAL) PEN MISSING FROM COST_NOW'
  STOP '(NODE_VAL)'
END IF
DO K = 1,NU
  IF (CU(K).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) CU(',K,') MISSING FROM COST_NOW'
    STOP '(NODE_VAL)'
  END IF
END DO
DO J = 1,NX
  IF (CX(J).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) CX(',J,') MISSING FROM COST_NOW'
    STOP '(NODE_VAL)'
  END IF
END DO
```

!
!
!

Check transistion function.

```
LEVEL = 1
Y = DUMMY
YU = DUMMY
YX = DUMMY
CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             S,Y,YU,YX,YW)
IF (LEVEL.NE.1) THEN
  WRITE (*,*) '(NODE_VAL) NOT ADAPTED FOR dy BY FINITE DIFF'
  STOP '(NODE_VAL)'
END IF
DO J = 1,NX
  IF (Y(J).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) Y(',J,') MISSING FROM TRANSX'
    STOP '(NODE_VAL)'
  END IF
END DO
```



```

DO K = 1,NU
  IF (YU(J,K).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) YU('',J,K,') MISSING FROM TRANSX'
    STOP '(NODE_VAL)'
  END IF
END DO
DO K = 1,NX
  IF (YX(J,K).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) YX('',J,K,') MISSING FROM TRANSX'
    STOP '(NODE_VAL)'
  END IF
END DO
DO K = 1,NW
  IF (YW(J,K).EQ.DUMMY) THEN
    WRITE (*,*) '(NODE_VAL) YW('',J,K,') MISSING FROM TRANSX'
    STOP '(NODE_VAL)'
  END IF
END DO
END DO

```

```

! -----
! Solve for current w.
! -----

```

```

IF (NEWTON) THEN
  LEVEL = LEVEL0
  FDIFF = .FALSE.
ELSE
  LEVEL = 0
  IF (LEVEL0.NE.0) FDIFF = .TRUE.
END IF
ISW = IW
ERRTEST = ERROR0

X0(:NX) = X
W0(:NW) = WHOLD
CALL OPT_SOLVt (LEVEL, U,F,FX,ERRTEST)

```

```

IF (ERRTEST) ERROR = .TRUE.

```

```

IF ( (NEWTON).AND.(LEVEL.NE.LEVEL0) ) THEN
  WRITE (*, '(A78)')
+   '(NODE_VAL) SOLVER DID NOT PROVIDE GRADIENTS, '//
+   'SWITCHING TO FINITE DIFFERENCES'
  FDIFF = .TRUE.
END IF

```

```

! -----
! Calculate derivative by finite differences
! if required but not provided by opt_solv.
! -----

```

```

! Save values.

```

```

UHOLD = U
FHOLD = F

```

```

IF (FDIFF) THEN

```

```

LEVEL = 0
DO J = 1,NX
  XDEL = X
  DELINV = DELINV2
  DO L = 1,2

```

```

! -----
! ID neighboring x used to calculate finite diff.
! Central diff approximation used unless outside
! the domain of x defined by the discretization.
! -----

```

```

SKIP = .FALSE.
IF (L.EQ.1) THEN
  XDEL(J) = X(J) - DEL
  IF (XDEL(J).LT.XBL(J)) THEN
    F1 = FHOLD
    U1 = UHOLD
    DELINV = DELINV1
    SKIP = .TRUE.
  END IF
ELSE
  XDEL(J) = X(J) + DEL
  IF (XDEL(J).GT.XBU(J)) THEN
    F2 = FHOLD
    U2 = UHOLD
    DELINV = DELINV1
    SKIP = .TRUE.
  END IF
END IF

```

```

! -----
! Solve for neighboring x if not outside domain.
! -----

```

```

IF (.NOT.SKIP) THEN

```

```

  Solve for xdel.

```

```

  X0(:NX) = XDEL
  W0(:NW) = WHOLD
  ERRTEST = ERROR0
  CALL OPT_SOLVt (LEVEL, U,F,FX,ERRTEST)

```

```

  Save values.

```

```

  IF (L.EQ.1) THEN
    F1 = F
    U1 = U
  ELSE
    F2 = F
    U2 = U
  END IF
END IF
END DO

```

```

! Calculate finite difference values.

```

```

      FX(J) = (F2 - F1)*DELINV
      UX(J,:) = (U2 - U1)*DELINV
END DO

!           Reset to initial solution.

      U = UHOLD
      F = FHOLD
      if (error) write (*,'(a26,50x,30f14.8)')
+         '(node_val) u,x,w,f,fx:',
+         u,x0(:NX),w0(:NW),f,fx
      END IF

!           -----
!           Get cost for current w.
!           -----

      SUMU = SUMU + U*PWN(IW)
      SUMF = SUMF + F*PWN(IW)
      IF (LEVEL0.EQ.1) SUMFX = SUMFX + FX*PWN(IW)

!           -----
!           Verify user supplied gradients.
!           -----

!           Check cost_now gradients.

      LEVEL = 1
      X = XHOLD
      W = WHOLD
      DO K = 1,NU
        U = UHOLD
        U(K) = UHOLD(K) - DEL
        CALL COST_NOW (LEVEL,NU,NX,NW,ISTAGE,IYEAR,ISEASON,U,X,W,
+          C1,CU,CX,PEN)
        U(K) = UHOLD(K) + DEL
        CALL COST_NOW (LEVEL,NU,NX,NW,ISTAGE,IYEAR,ISEASON,U,X,W,
+          C2,CU,CX,PEN)
        G = (C2 - C1)*DELINV2
        ERRST = ERRTOL*(1.0 + ABS(C1) + ABS(G))
        IF (ABS(CU(K)-G).GT.ERRST) THEN
          WRITE (*,*) '(NODE_VAL) CU <> G',CU(K),G,CU(K)-G
          WRITE (*,*) 'IX,IW,K:',IXNODE,IW,K
          WRITE (*,*) 'ISTAGE,IYEAR,ISEASON:',ISTAGE,IYEAR,ISEASON
          WRITE (*,*) 'C1,C2,DEL:', C1, C2, DEL
          WRITE (*,*) 'U:', UHOLD
          WRITE (*,*) 'X:', X
          WRITE (*,*) 'W:', W
          STOP '(NODE_VAL)'
        END IF
      END DO

      U = UHOLD
      W = WHOLD
      DO K = 1,NX
        X = XHOLD(:NX)
        X(K) = XHOLD(K) - DEL
        CALL COST_NOW (LEVEL,NU,NX,NW,ISTAGE,IYEAR,ISEASON,U,X,W,

```

```

+           C1,CU,CX,PEN)
X(K) = XHOLD(K) + DEL
CALL COST_NOW (LEVEL,NU,NX,NW,ISTAGE,IYEAR,ISEASON,U,X,W,
+           C2,CU,CX,PEN)
G = (C2 - C1)*DELINV2
ERRTST = ERRTOL*(1.0 + ABS(C1) + ABS(G))
IF (ABS(CX(K)-G).GT.ERRTST) THEN
  WRITE (*,*) '(NODE_VAL) CX <> G',CX(K),G,CX(K)-G
  WRITE (*,*) 'IX,IW,K:',IXNODE,IW,K
  WRITE (*,*) 'ISTAGE,IYEAR,ISEASON:',ISTAGE,IYEAR,ISEASON
  WRITE (*,*) 'C1,C2,DEL:', C1, C2, DEL
  WRITE (*,*) 'U:', U
  WRITE (*,*) 'X:', XHOLD
  WRITE (*,*) 'W:', W
  STOP '(NODE_VAL)'
END IF
END DO

```

! Check transX gradients.

```

LEVEL = 1
X = XHOLD
W = WHOLD
DO K = 1,NU
  U = UHOLD
  U(K) = UHOLD(K) - DEL
  CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             S,Y1,YU,YX,YW)
  U(K) = UHOLD(K) + DEL
  CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             S,Y2,YU,YX,YW)
DO J = 1,NX
  G = (Y2(J) - Y1(J))*DELINV2
  ERRTST = ERRTOL*(1.0 + ABS(Y1(J)) + ABS(G))
  IF (ABS(YU(J,K)-G).GT.ERRTST) THEN
    WRITE (*,*) '(NODE_VAL) YU <> G',YU(J,K),G,YU(J,K)-G
    WRITE (*,*) 'IX,IW,J,K:',IXNODE,IW,J,K
    WRITE (*,*) 'ISTAGE,IYEAR,ISEASON:',ISTAGE,IYEAR,ISEASON
    WRITE (*,*) 'Y1,Y2,DEL:', Y1(J), Y2(J), DEL
    WRITE (*,*) 'U:', UHOLD
    WRITE (*,*) 'X:', X
    WRITE (*,*) 'W:', W
    STOP '(NODE_VAL)'
  END IF
END DO
END DO

U = UHOLD
W = WHOLD
DO K = 1,NX
  X = XHOLD
  X(K) = XHOLD(K) - DEL
  CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+             ISTAGE,IYEAR,ISEASON,U,X,W,
+             S,Y1,YU,YX,YW)
  X(K) = XHOLD(K) + DEL

```

```

      CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+               ISTAGE,IYEAR,ISEASON,U,X,W,
+               S,Y2,YU,YX,YW)
      DO J = 1,NX
        G = (Y2(J) - Y1(J))*DELINV2
        ERRTST = ERRTOL*(1.0 + ABS(Y1(J)) + ABS(G))
        IF (ABS(YX(J,K)-G).GT.ERRTST) THEN
          WRITE (*,*) '(NODE_VAL) YX <> G',YX(J,K),G,YX(J,K)-G
          WRITE (*,*) 'IX,IW,J,K:',IXNODE,IW,J,K
          WRITE (*,*) 'ISTAGE,IYEAR,ISEASON:',ISTAGE,IYEAR,ISEASON
          WRITE (*,*) 'Y1,Y2,DEL:', Y1(J), Y2(J), DEL
          WRITE (*,*) 'U:', U
          WRITE (*,*) 'X:', XHOLD
          WRITE (*,*) 'W:', W
          STOP '(NODE_VAL)'
        END IF
      END DO
    END DO

    U = UHOLD
    X = XHOLD
    DO K = 1,NW
      W = WHOLD
      W(K) = WHOLD(K) - DEL
      CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+               ISTAGE,IYEAR,ISEASON,U,X,W,
+               S,Y1,YU,YX,YW)
      W(K) = WHOLD(K) + DEL
      CALL TRANSXt (LEVEL,NU,NX,NW,NTNLN,
+               ISTAGE,IYEAR,ISEASON,U,X,W,
+               S,Y2,YU,YX,YW)
    DO J = 1,NX
      G = (Y2(J) - Y1(J))*DELINV2
      ERRTST = ERRTOL*(1.0 + ABS(Y1(J)) + ABS(G))
      IF (ABS(YW(J,K)-G).GT.ERRTST) THEN
        WRITE (*,*) '(NODE_VAL) YW <> G',YW(J,K),G,YW(J,K)-G
        WRITE (*,*) 'IX,IW,J,K:',IXNODE,IW,J,K
        WRITE (*,*) 'ISTAGE,IYEAR,ISEASON:',ISTAGE,IYEAR,ISEASON
        WRITE (*,*) 'Y1,Y2,DEL:', Y1(J), Y2(J), DEL
        WRITE (*,*) 'U:', U
        WRITE (*,*) 'X:', X
        WRITE (*,*) 'W:', WHOLD
        STOP '(NODE_VAL)'
      END IF
    END DO
  END DO

END IF
END DO

U = SUMU
F = SUMF
FX = SUMFX
LEVEL = LEVEL0

```

```

!
!
!

```

```

-----
Verify outputs.
-----

```

```

NT_INT = N_INT - NT_INT
NT_OBJ = N_OBJ - NT_OBJ
NT_OPT = N_OPT - NT_OPT
NT_SOL = N_SOL - NT_SOL

```

END

Subroutine ADJ_MOD

This routine adjusts cost-to-go values and gradients to ensure convex interpolation. It is assumed that if node values are consistent with a convex function, then the interpolation should try to preserve this convexity. This is accomplished by adjusting the gradients and, if necessary, the cost-to-go to satisfy the one-dimensional convexity constraints of Chapter Five. This prevents small oscillations or numerical error from creating false local minima that become significant with additional recursions (i.e., stages).

```

SUBROUTINE ADJ_MOD ( ISTAGE, IYEAR, ISEASON, IBELOW, IABOVE,
+                   XN, FN, FXN )

IMPLICIT NONE
INCLUDE 'I.SIZEPROB'      !Parameters for problem size.
INTEGER                  ISTAGE, IYEAR, ISEASON,
+                        IBELOW(NX,NNODES), IABOVE(NX,NNODES)
DOUBLE PRECISION         XN(NX,NNODES),
+                        FN(NNODES), FXN(NX,NNODES)

!-----
!Adjusts cost function solutions using user-supplied constraints, and,
! if gdp, then adjusts gradients to produce a convex Hermite interpolat.
!-----
!ix = 1 is assumed to be the lowest node in the domain.
!Each node ix has the associated characteristics:
!           xn      location of node
!           fn      function value at node
!           fxn     function gradient at node
!Nodes and values are contained in a linked list that points to nodes
! immediately above and below in each dimension. The following are
! special cases:
!--Nodes on the edge of the domain:          an adjacent node will be 0
!--Interpolated nodes used for continuity:   an adjacent node will be -1
!G used for finite difference gradients.
!-----

!           Local variables for specf.

DOUBLE PRECISION         FMIN, FMAX, FXMIN(NX), FXMAX(NX)

!           Other local variables.

INTEGER                  MAXITER, MIDITER, NITER,
+                        IX, J, IXB, IXA, KEY, IXBB, IXAA

```

```

DOUBLE PRECISION      ERRTOL, TOL, FNEW, ACCEL,
+                     F0(NNODES), FX0(NX,NNODES),
+                     FHOLD(NNODES), FXHOLD(NX,NNODES),
+                     X, XB, XA, F, FB, FA, FX, FXB, FXA, GB, GA,
+                     GA3, GAMIN, GAMAX, GB3, GBMIN, GBMAX,
+                     D, DA, DB, DENOM, FRAC,
+                     FXCA1, FXCA2, FXCB1, FXCB2,
+                     XBB, XAA, GBB, GAA,
+                     FDIFF, FXDIFF(NX)
LOGICAL               CONVEX(NX,NNODES),
+                     NOCHANGE, DONE,
+                     CA1, CA2, CB1, CB2, FINALTRY, SKIP

EXTERNAL              SPECF

!-----

ERRTOL = 1.0E-15
MAXITER = 21
ACCEL = 2.0

!           Set end-of-sharing iteration at which entire
!           adjustment is taken by current node, if possible.

MIDITER = 6

!           -----
!           Save inputs.
!           -----

WRITE (*,*) '(ADJ_MOD) BEGIN, STAGE =', ISTAGE

F0 = FN
FX0 = FXN

!           -----
!           Get bounds on function values.
!           -----

FMIN = -1.0E20
FMAX = 1.0E20
FXMIN = -1.0E20
FXMAX = 1.0E20

CALL SPECF (NX,ISTAGE,IYEAR,ISEASON, FMIN,FMAX,FXMIN,FXMAX)

!           Verify parameters specified correctly.

IF (FMAX.LT.FMIN) THEN
  WRITE (*,*) '(MODELSTG) FMAX < FMIN', FMAX, FMIN, FMAX-FMIN
  STOP '(MODELSTG)'
END IF
DO J = 1,NX
  IF (FXMAX(J).LT.FXMIN(J)) THEN
    WRITE (*,*) '(MODELSTG) FXMAX < FXMIN', J, FXMAX(J), FXMIN(J)
    STOP '(MODELSTG)'
  END IF
  IF ( (FMIN.NE.-1.0E20).OR.(FMAX.NE.1.0E20) ) THEN
    IF ( (FXMIN(J).GT.0.0).OR.(FXMAX(J).LT.0.0) ) THEN

```

```

        WRITE (*,*) '(MODELSTG) FX = 0.0 MUST BE WITHIN BOUNDS'
        WRITE (*,*) '        CHANGE FXMIN,FXMAX:', FXMIN,FXMAX
        STOP '(MODELSTG)'
    END IF
END IF
END DO

```

```

!-----
!Adjust function values.
!-----

```

```

!-----
!Adjust consistent with specf.
!-----

```

```

DO IX = 1,NNODES
    F = FN(IX)
    IF (F.LT.FMIN) THEN
        WRITE (*, '(A20,I6,2F16.8,E8.2)')
+       '(ADJ_MOD) F < FMIN', IX, F, FMIN, F-FMIN
        F = FMIN
    ELSE IF (F.GT.FMAX) THEN
        WRITE (*, '(A20,I6,2F16.8,E8.2)')
+       '(ADJ_MOD) F > FMAX', IX, F, FMAX, F-FMAX
        F = FMAX
    END IF
    FN(IX) = F
END DO

```

```

!-----
!Write non-convex nodes grouped by dimension.
!-----

```

```

DO J = 1,NX
    DO IX = 1,NNODES
        IXB = IBELOW(J,IX)
        IXA = IABOVE(J,IX)
        IF ( (IXB.GT.0).AND.(IXA.GT.0) ) THEN

            F = FN(IX)
            X = XN(J,IX)
            XB = XN(J,IXB)
            FB = FN(IXB)
            XA = XN(J,IXA)
            FA = FN(IXA)

            IF (XB.GE.X) THEN
                WRITE (*,*) '(ADJ_MOD) ERROR 1 DIVIDE', XB, X
                STOP '(ADJ_MOD)'
            END IF
            IF (X.GE.XA) THEN
                WRITE (*,*) '(ADJ_MOD) ERROR 2 DIVIDE', X, XA
                STOP '(ADJ_MOD)'
            END IF

            GB = (F - FB)/(X - XB)
            GA = (FA - F)/(XA - X)

```



```

TOL = ERRTOL*(1.0 + ABS(GA) + ABS(GB))
TOL = MIN( TOL,1.0 )
IF (GB-TOL.GT.GA) THEN
  WRITE (*,'(A53,I6,I4,2F14.8,E10.4,3F12.4,20F10.2)')
+   '(ADJ_MOD) ***NOT CONVEX*** IX,J,GB>GA,DIFF,B/F/A,X:',
+   IX,J,GB,GA,GB-GA,FB,F,FA,XN(:,IX)
  END IF
END IF
END DO
END DO

```

```

! -----
! Adjust values for convexity.
! -----

```

```

! Assumes lack of convexity due to error in interpol.
! Values adjusted iteratively using adjacent nodes.

```

```

NITER = 0
DO
  NITER = NITER + 1
  IF (P) WRITE (*,*) '(ADJ_MOD) BEGINING VALUE ITERATION', NITER
  NOCHANGE = .TRUE.
  FHOLD = FN

  DO IX = 1,NNODES
    F = FHOLD(IX)
    DO J = 1,NX
      IXB = IBELOW(J,IX)
      IXA = IABOVE(J,IX)
      IF ( (IXB.GT.0).AND.(IXA.GT.0) ) THEN
        X = XN(J,IX)
        XB = XN(J,IXB)
        FB = FHOLD(IXB)
        XA = XN(J,IXA)
        FA = FHOLD(IXA)

        FNEW = ( FB*(XA-X) + FA*(X-XB) ) / (XA-XB)
        IF (FNEW.LT.FHOLD(IX)) THEN
          NOCHANGE = .FALSE.
          F = MIN( F,FNEW )
        END IF
      END IF
    END DO
    IF (F.LT.FMIN) THEN
      NOCHANGE = .FALSE.
      F = FMIN
    END IF
    FN(IX) = F
  END DO

  IF (NOCHANGE) THEN
    EXIT
  ELSE IF (NITER.GE.MAXITER*20) THEN
    WRITE (*,*) '(ADJ_MOD) WARNING: MAX ITERATIONS EXCEEDED'
    EXIT
  END IF

```

END DO

```
!-----
! Write changed nodes.
!-----
```

```
DO IX = 1,NNODES
  FDIFF = FN(IX) - F0(IX)
  TOL = ERRTOL*(1.0 + ABS(FN(IX)) + ABS(F0(IX)))
  TOL = MIN( TOL,1.0 )
  IF (ABS(FDIFF).GT.TOL)
+   write (*,'(a33,3x,i6,3f12.4,40f10.2)')
+       '(adj_mod) ix,f(old/diff/new),x:',
+       ix,f0(ix),fdiff,fn(ix),xn(:,ix)
END DO
```

```
!-----
! Verify adjusted values consistent with convex func.
!-----
```

```
CONVEX = .TRUE.
DO IX = 1,NNODES
  DO J = 1,NX
    IXB = IBELOW(J,IX)
    IXA = IABOVE(J,IX)
    IF ( (IXB.GT.0).AND.(IXA.GT.0) ) THEN
      X = XN(J,IX)
      F = FN(IX)
      XB = XN(J,IXB)
      FB = FN(IXB)
      XA = XN(J,IXA)
      FA = FN(IXA)

      GB = (F - FB)/(X - XB)
      GA = (FA - F)/(XA - X)

      IF (GB.GT.GA) THEN
        CONVEX(J,IX) = .FALSE.
        TOL = ERRTOL*(1.0 + ABS(GA) + ABS(GB))
        TOL = MIN( TOL,1.0 )
        IF (GB-TOL.GT.GA) THEN
+          WRITE (*,'(A53,I6,I4,2F14.8,E10.4,3F12.4,20F10.2)')
+          '(ADJ_MOD) ***NOT CONVEX*** IX,J,GB>GA,DIFF,B/F/A,X:',
+          IX,J,GB,GA,GB-GA,FB,F,FA,XN(:,IX)
        END IF
      END IF
    END IF
  END DO
END DO
```

```
!-----
! Adjust function gradients.
!-----
```

```
!-----
! Adjust consistent with specf.
!-----
```

```

DO IX = 1,NNODES
  IF ( (FN(IX).EQ.FMIN).OR.(FN(IX).EQ.FMAX) ) FXN(:,IX) = 0.0
  DO J = 1,NX
    FX = FXN(J,IX)
    IF (FX.LT.FXMIN(J)) THEN
      IF (FX+ERRTOL.LT.FXMIN(J)) THEN
        WRITE (*,'(A22,I6,I4,2F14.8,E10.2)')
+       '(ADJ_MOD) FX < FXMIN',IX,J,FX,FXMIN(J),FX-FXMIN(J)
        END IF
        FX = FXMIN(J)
      ELSE IF (FX.GT.FXMAX(J)) THEN
        IF (FX-ERRTOL.GT.FXMAX(J)) THEN
          WRITE (*,'(A22,I6,I4,2F14.8,E10.2)')
+          '(ADJ_MOD) FX > FXMAX',IX,J,FX,FXMAX(J),FX-FXMAX(J)
          END IF
          FX = FXMAX(J)
        END IF
        FXN(J,IX) = FX
      END DO
    END DO
  END DO

! -----
! Calc. finite diffs and verify consistent with specf.
! -----

DO IX = 1,NNODES
  F = FN(IX)
  DO J = 1,NX
    X = XN(J,IX)

    IXB = IBELOW(J,IX)
    IF (IXB.GT.0) THEN
      XB = XN(J,IXB)
      FB = FN(IXB)
      GB = (F - FB)/(X - XB)
      TOL = ERRTOL*(1.0 + ABS(GB))
      TOL = MIN( TOL,1.0 )
      IF (GB+TOL.LT.FXMIN(J))
+      WRITE (*,'(A26,I6,I4,2F14.8,E10.2)')
+      '(ADJ_MOD) ERROR: GB < MIN',IX,J,GB,FXMIN(J),GB-FXMIN(J)
    END IF

    IXA = IABOVE(J,IX)
    IF (IXA.GT.0) THEN
      XA = XN(J,IXA)
      FA = FN(IXA)
      GA = (FA - F)/(XA - X)
      TOL = ERRTOL*(1.0 + ABS(GA))
      TOL = MIN( TOL,1.0 )
      IF (GA-TOL.GT.FXMAX(J))
+      WRITE (*,'(A26,I6,I4,2F14.8,E10.2)')
+      '(ADJ_MOD) ERROR: GA > MAX',IX,J,GA,FXMAX(J),GA-FXMAX(J)
    END IF
  END DO
END DO

```

! -----

```

!           Adjust gradients when using Hermite interpolation.
!           -----

!           Note:  since no absolute reference, adjust values
!           only at current node and let values at adjacent
!           nodes be adjusted in their turn.

FINALTRY = .FALSE.
IF (GDP) THEN
  NITER = 0
  DO
! -----
    NITER = NITER + 1
    WRITE (*,*) '(ADJ_MOD) BEGINING GRADIENT ITERATION', NITER
    DONE = .TRUE.
    FXHOLD = FXN
! -----

!           -----
!           Adjust gradients consistent with convex function.
!           -----

DO IX = 1,NNODES
  F = FN(IX)
  DO J = 1,NX

!           Get useful values.

    X = XN(J,IX)
    FX = FXHOLD(J,IX)

    IXB = IBELOW(J,IX)
    IF (IXB.GT.0) THEN
      XB = XN(J,IXB)
      FB = FN(IXB)
      FXB = FXHOLD(J,IXB)
      GB = (F - FB)/(X - XB)
    END IF

    IXA = IABOVE(J,IX)
    IF (IXA.GT.0) THEN
      XA = XN(J,IXA)
      FA = FN(IXA)
      FXA = FXHOLD(J,IXA)
      GA = (FA - F)/(XA - X)
    END IF

!           Adjust lower bound gradient.

    IF (IXB.LE.0) THEN
      IF (FX.GT.GA) THEN
        DONE = .FALSE.
        TOL = ERRTOL*(1.0 + ABS(GA))
        TOL = MIN( TOL,1.0 )
        IF ( P.AND.(FX-TOL.GT.GA) )
+          WRITE (*, '(A22,I6,I4,2F14.8,E10.2)')
+          '(ADJ_MOD) FX > GA ',IX,J,FX, GA,FX-GA
        IF (FXA.GT.GA) THEN
          FX = GA - TOL

```

```

        ELSE
            FX = GA
        END IF
    END IF

!           Adjust upper bound gradient.

ELSE IF (IXA.LE.0) THEN
    IF (FX.LT.GB) THEN
        DONE = .FALSE.
        TOL = ERRTOL*(1.0 + ABS(GB))
        TOL = MIN( TOL,1.0 )
        IF ( P.AND.(FX+TOL.LT.GB) )
+           WRITE (*,'(A22,I6,I4,2F14.8,E10.2)')
+           '(ADJ_MOD) FX < GB ',IX,J,FX, GB,FX-GB
        IF (FXB.LT.GB) THEN
            FX = GB + TOL
        ELSE
            FX = GB
        END IF
    END IF

!           Adjust internal gradient.
!           No change if finite diff indicates not convex.

ELSE IF (CONVEX(J,IX)) THEN
    IF (GB.GT.GA) THEN
        WRITE (*,*) '(ADJ_MOD) ERROR 1:', J,IX,FB,F,FA,GB,GA
        STOP '(ADJ_MOD)'
    END IF

    TOL = ERRTOL*(1.0 + ABS(GA) + ABS(GB))
    TOL = MIN( TOL,1.0 )

    IF (FX.GE.GB) THEN
        IF (FX.GT.GA) THEN
            DONE = .FALSE.
            IF ( P.AND.(FX-TOL.GT.GA) )
+               WRITE (*,'(A22,I6,I4,2F14.8,E10.2)')
+               '(ADJ_MOD) FX > GA ',IX,J, FX,GA, FX-GA
            IF (GB+(TOL+TOL).LT.GA) THEN
                FX = GA - TOL
            ELSE
                FX = 0.5*(GB + GA)
            END IF
        END IF

    ELSE IF (FX.LE.GA) THEN
        IF (FX.LT.GB) THEN
            DONE = .FALSE.
            IF ( P.AND.(FX+TOL.LT.GB) )
+               WRITE (*,'(A22,I6,I4,2F14.8,E10.2)')
+               '(ADJ_MOD) FX < GB ',IX,J, FX,GB ,FX-GB
            IF (GB+(TOL+TOL).LT.GA) THEN
                FX = GB + TOL
            ELSE
                FX = 0.5*(GB + GA)
            END IF
        END IF
    END IF

```

```

        END IF

    ELSE
        FX = 0.5*(GB + GA)
        IF (GB-TOL.GT.GA) THEN
            WRITE (*,*) '(ADJ_MOD) ERROR:  GB>GA',IX,J,GB,GA,FX
            STOP '(ADJ_MOD)'
        END IF
    END IF
END IF

FXN(J,IX) = FX

END DO
END DO

!
! -----
! Adjust gradients for convex function interpolation.
! -----
!
! Adjustments to gradients at both ends of an interval
! are made in proportion to their deviation from the
! finite difference gradient.  The combined
! adjustment on both nodes should result in convex
! interpolation.
! Tight is passed in common.
!

FXHOLD = FXN
DO IX = 1,NNODES
    F = FN(IX)
    DO J = 1,NX
        SKIP = .FALSE.

!           Get useful values.

        X = XN(J,IX)
        FX = FXHOLD(J,IX)

        IXB = IBELOW(J,IX)
        IF (IXB.GT.0) THEN
            XB = XN(J,IXB)
            FB = FN(IXB)
            FXB = FXHOLD(J,IXB)
            GB = (F - FB)/(X - XB)
            GB3 = (3.0 + TIGHT + TIGHT)*GB
            GBMAX = (1.0 + TIGHT)*FXB + (2.0 + TIGHT)*FX
            GBMIN = (2.0 + TIGHT)*FXB + (1.0 + TIGHT)*FX
        END IF

        IXA = IABOVE(J,IX)
        IF (IXA.GT.0) THEN
            XA = XN(J,IXA)
            FA = FN(IXA)
            FXA = FXHOLD(J,IXA)
            GA = (FA - F)/(XA - X)
            GA3 = (3.0 + TIGHT + TIGHT)*GA
            GAMAX = (1.0 + TIGHT)*FX + (2.0 + TIGHT)*FXA
            GAMIN = (2.0 + TIGHT)*FX + (1.0 + TIGHT)*FXA
        END IF
    END DO
END DO

```

END IF

! Adjust lower bound gradient.

```
IF (IXB.LE.0) THEN
  TOL = ERRTOL*(1.0 + ABS(GA3))
  TOL = MIN( TOL,1.0 )
  IF (GAMIN-TOL.GT.GA3) THEN
    DONE = .FALSE.
    D = GA - FX
    DA = FXA - GA
    IF (D.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE D (1)', IX,J,D
      STOP '(ADJ_MOD)'
    END IF
    IF (DA.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE DA (1)', IX,J,DA
      SKIP = .TRUE.
    END IF
    IF (NITER.LT.MIDITER) THEN
      DENOM = (2.0 + TIGHT)*D + (1.0 + TIGHT)*DA
    ELSE
      DENOM = (2.0 + TIGHT)*D
    END IF
    IF (ABS(DENOM).LE.TOL) THEN
      FX = GA
      KEY = 81
    ELSE
      IF (DENOM.GT.0.0) THEN
        FRAC = (GAMIN - GA3)/DENOM
        FX = GA - D*(1.0 + FRAC)
      ELSE
        FX = GA
      END IF
      KEY = 1
    END IF
  ELSE IF (GAMAX+TOL.LT.GA3) THEN
    DONE = .FALSE.
    D = GA - FX
    DA = FXA - GA
    IF (D.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE D (2)', IX,J,D
      STOP '(ADJ_MOD)'
    END IF
    IF (DA.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE DA (2)', IX,J,DA
      SKIP = .TRUE.
    END IF
    IF (NITER.LT.MIDITER) THEN
      DENOM = (1.0 + TIGHT)*D + (2.0 + TIGHT)*DA
    ELSE
      DENOM = (1.0 + TIGHT)*D
    END IF
    IF (ABS(DENOM).LE.TOL) THEN
      FX = GA
      KEY = 82
    ELSE
      IF (DENOM.GT.0.0) THEN
```

```

        FRAC = (GA3 - GAMAX)/DENOM
        FX = GA - D*(1.0 - FRAC)
    ELSE
        FX = GA
    END IF
    KEY = 2
END IF
    if (p) write (*,'(a35,3i6,8x,3f22.16,10f9.2)')
+       '(adj_mod) key,ix,j,f,fxhold,fx,x:',
+       key,ix,j,f,fxhold(j,ix),fx,xn(:,ix)
END IF

!           Adjust upper bound gradient.

ELSE IF (IXA.LE.0) THEN
    TOL = ERRTOL*(1.0 + ABS(GB3))
    TOL = MIN( TOL,1.0 )
    IF (GBMIN-TOL.GT.GB3) THEN
        DONE = .FALSE.
        DB = GB - FXB
        D = FX - GB
        IF (D.LT.0.0) THEN
            WRITE (*,*) 'NEGATIVE D (3)', IX,J,D
            STOP '(ADJ_MOD)'
        END IF
        IF (DB.LT.0.0) THEN
c           WRITE (*,*) 'NEGATIVE DB (3)', IX,J,DB
            SKIP = .TRUE.
        END IF
        IF (NITER.LT.MIDITER) THEN
            DENOM = (2.0 + TIGHT)*DB + (1.0 + TIGHT)*D
        ELSE
            DENOM = (1.0 + TIGHT)*D
        END IF
        IF (ABS(DENOM).LE.TOL) THEN
            FX = GB
            KEY = 83
        ELSE
            IF (DENOM.GT.0.0) THEN
                FRAC = (GBMIN - GB3)/DENOM
                FX = GB + D*(1.0 - FRAC)
            ELSE
                FX = GB
            END IF
            KEY = 3
        END IF
    ELSE IF (GBMAX+TOL.LT.GB3) THEN
        DONE = .FALSE.
        DB = GB - FXB
        D = FX - GB
        IF (D.LT.0.0) THEN
            WRITE (*,*) 'NEGATIVE D (4)', IX,J,D
            STOP '(ADJ_MOD)'
        END IF
        IF (DB.LT.0.0) THEN
c           WRITE (*,*) 'NEGATIVE DB (4)', IX,J,DB
            SKIP = .TRUE.
        END IF
    END IF

```



```

      IF (NITER.LT.MIDITER) THEN
        DENOM = (1.0 + TIGHT)*DB + (2.0 + TIGHT)*D
      ELSE
        DENOM = (2.0 + TIGHT)*D
      END IF
      IF (ABS(DENOM).LE.TOL) THEN
        FX = GB
        KEY = 84
      ELSE
        IF (DENOM.GT.0.0) THEN
          FRAC = (GB3 - GBMAX)/DENOM
          FX = GB + D*(1.0 + FRAC)
        ELSE
          FX = GB
        END IF
        KEY = 4
      END IF
    END IF
  END IF

```

```

!           Adjust internal gradient.
!           If cannot id satisfactory gradient, use reasonable
!           value, and let other values adjust in looping.

```

```

ELSE IF (CONVEX(J,IX)) THEN
  IF (GB.GT.GA) THEN
    WRITE (*,*) '(ADJ_MOD) ERROR 2:', J,IX,FB,F,FA,GB,GA
    STOP '(ADJ_MOD)'
  END IF

```

```

TOL = ERRTOL*(1.0 + ABS(GA3) + ABS(GB3))
TOL = MIN( TOL,1.0 )

```

```

IF (GAMAX+TOL.LT.GA3) THEN
  CA2 = .TRUE.
  D = GA - FX
  DA = FXA - GA
  IF (D.LT.0.0) THEN
    WRITE (*,*) 'NEGATIVE D (5)', IX,J,D
    STOP '(ADJ_MOD)'
  END IF
  IF (DA.LT.0.0) THEN
    WRITE (*,*) 'NEGATIVE DA (5)', IX,J,DA
    SKIP = .TRUE.
  END IF
  IF (NITER.LT.MIDITER) THEN
    DENOM = (1.0 + TIGHT)*D + (2.0 + TIGHT)*DA
  ELSE
    DENOM = (1.0 + TIGHT)*D
  END IF
  IF (ABS(DENOM).LE.TOL) THEN
    FXCA2 = GA
  ELSE
    IF (DENOM.GT.0.0) THEN
      FRAC = (GA3 - GAMAX)/DENOM
      FXCA2 = GA - D*(1.0 - FRAC)
    ELSE
      FXCA2 = GA
    END IF
  END IF

```

c

```

      END IF
    ELSE
      CA2 = .FALSE.
      FXCA2 = (GA3 - FXA*(2.0 + TIGHT))/(1.0 + TIGHT)
    END IF

    IF (GAMIN-TOL.GT.GA3) THEN
      CA1 = .TRUE.
      D = GA - FX
      DA = FXA - GA
      IF (D.LT.0.0) THEN
        WRITE (*,*) 'NEGATIVE D (6)', IX,J,D
        STOP '(ADJ_MOD)'
      END IF
      IF (DA.LT.0.0) THEN
        WRITE (*,*) 'NEGATIVE DA (6)', IX,J,DA
        SKIP = .TRUE.
      END IF
      IF (NITER.LT.MIDITER) THEN
        DENOM = (2.0 + TIGHT)*D + (1.0 + TIGHT)*DA
      ELSE
        DENOM = (2.0 + TIGHT)*D
      END IF
      IF (ABS(DENOM).LE.TOL) THEN
        FXCA1 = GA
      ELSE
        IF (DENOM.GT.0.0) THEN
          FRAC = (GAMIN - GA3)/DENOM
          FXCA1 = GA - D*(1.0 + FRAC)
        ELSE
          FXCA1 = GA
        END IF
      END IF
    ELSE
      CA1 = .FALSE.
      FXCA1 = (GA3 - FXA*(1.0 + TIGHT))/(2.0 + TIGHT)
    END IF

    IF (GBMAX+TOL.LT.GB3) THEN
      CB2 = .TRUE.
      DB = GB - FXB
      D = FX - GB
      IF (D.LT.0.0) THEN
        WRITE (*,*) 'NEGATIVE D (7)', IX,J,D
        STOP '(ADJ_MOD)'
      END IF
      IF (DB.LT.0.0) THEN
        WRITE (*,*) 'NEGATIVE DB (7)', IX,J,DB
        SKIP = .TRUE.
      END IF
      IF (NITER.LT.MIDITER) THEN
        DENOM = (1.0 + TIGHT)*DB + (2.0 + TIGHT)*D
      ELSE
        DENOM = (2.0 + TIGHT)*D
      END IF
      IF (ABS(DENOM).LE.TOL) THEN
        FXCB2 = GB
      ELSE

```

```

      IF (DENOM.GT.0.0) THEN
        FRAC = (GB3 - GBMAX)/DENOM
        FXCB2 = GB + D*(1.0 + FRAC)
      ELSE
        FXCB2 = GB
      END IF
    END IF
  ELSE
    CB2 = .FALSE.
    FXCB2 = (GB3 - FXB*(1.0 + TIGHT))/(2.0 + TIGHT)
  END IF

  IF (GBMIN-TOL.GT.GB3) THEN
    CB1 = .TRUE.
    DB = GB - FXB
    D = FX - GB
    IF (D.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE D (8)', IX,J,D
      STOP '(ADJ_MOD)'
    END IF
    IF (DB.LT.0.0) THEN
      WRITE (*,*) 'NEGATIVE DB (8)', IX,J,DB
      SKIP = .TRUE.
    END IF
    IF (NITER.LT.MIDITER) THEN
      DENOM = (2.0 + TIGHT)*DB + (1.0 + TIGHT)*D
    ELSE
      DENOM = (1.0 + TIGHT)*D
    END IF
    IF (ABS(DENOM).LE.TOL) THEN
      FXCB1 = GB
    ELSE
      IF (DENOM.GT.0.0) THEN
        FRAC = (GBMIN - GB3)/DENOM
        FXCB1 = GB + D*(1.0 - FRAC)
      ELSE
        FXCB1 = GB
      END IF
    END IF
  ELSE
    CB1 = .FALSE.
    FXCB1 = (GB3 - FXB*(2.0 + TIGHT))/(1.0 + TIGHT)
  END IF

```

```

!           If finaltry, adjustments are not bounded to
!           prevent non-convex interpolation in neighboring
!           domains (in order to allow adjustments to
!           propagate through domains that are ok).
!           However, adjustments must still be bounded
!           by finite-difference gradient of these neighboring
!           domains.

```

```

  IF (FINALTRY) THEN
    IF (CA2) THEN
      IXAA = IABOVE(J,IXA)
      IF ( (IXAA.GT.0).AND.(CONVEX(J,IXA)) ) THEN
        XAA = XN(J,IXAA)
        IF (XA.GE.XAA) THEN

```

```

        WRITE (*,*) '(ADJ_MOD) ERROR 3 DIVIDE', XA, XAA
        STOP '(ADJ_MOD)'
    END IF
    GAA = (FN(IXAA) - FA)/(XN(J,IXAA) - XA)
    FXCA2 = MIN(FXCA2,GAA)
END IF
END IF

IF (CA1) FXCA1 = MAX(FXCA1,GB)

IF (CB2) FXCB2 = MIN(FXCB2,GA)

IF (CB1) THEN
    IXBB = IBELOW(J,IXB)
    IF ( (IXBB.GT.0).AND.(CONVEX(J,IXB)) ) THEN
        XBB = XN(J,IXBB)
        IF (XBB.GE.XB) THEN
            WRITE (*,*) '(ADJ_MOD) ERROR 4 DIVIDE', XBB, XB
            STOP '(ADJ_MOD)'
        END IF
        GBB = (FB - FN(IXBB))/(XB - XBB)
        FXCB1 = MAX(FXCB1,GBB)
    END IF
END IF
END IF
END IF

!
!       Note:  adjustment for violation in either pair
!       of constraints [cb1,ca2] or [cb2,ca1] can cause
!       violation of other constraint in pair.
!       Note that constraint pairs [cb1,cb2] or [ca1,ca2]
!       cannot be violated at same time.
!       Also, if constraint pairs [cb1,ca1] or [cb2,ca2]
!       are violated together, adjustment for one
!       constraint will be in correct direction for other.
!

IF ( (CB2.OR.CA1).AND.(.NOT.SKIP) ) THEN
    DONE = .FALSE.
    IF (FXCB2+TOL.LT.FXCA1) THEN          !no problem with conflict
        IF (CB2) FX = FXCB2
        IF (CA1) FX = FXCA1
        IF (CB2.AND.CA1) THEN              !values close.
            WRITE (*,*) '(ADJ_MOD) WARNING CB2 & CA1, IX,J=',IX,J
            WRITE (*,*) 'FXCB2,FXCA1', FXCB2,FXCA1
            WRITE (*,*) 'FXCB1,FXCA2', FXCB1,FXCA2
            WRITE (*,*) 'IXB,XB,FB,FXB', IXB,XB,FB,FXB
            WRITE (*,*) 'IX, X, F, FX ', IX, X, F, FXHOLD(J,IX)
            WRITE (*,*) 'IXA,XA,FA,FXA', IXA,XA,FA,FXA
            WRITE (*,*) 'GB,GB3', GB,GB3
            WRITE (*,*) 'GBMIN,GBMAX', GBMIN,GBMAX
            WRITE (*,*) 'GA,GA3', GA,GA3
            WRITE (*,*) 'GAMIN,GAMAX', GAMIN,GAMAX
        END IF
        KEY = 11
    ELSE
        IF (.NOT.CB2) THEN
            FX = FXCB2
            KEY = 12
            IF (FINALTRY) THEN

```

```

        FX = FXCA1
        KEY = 22
    END IF
ELSE IF (.NOT.CA1) THEN
    FX = FXCA1
    KEY = 13
    IF (FINALTRY) THEN
        FX = FXCB2
        KEY = 23
    END IF
ELSE IF (FXCB2-TOL.GT.FXCA1) THEN
    DA = GA - FXCA1
    DB = FXCB2 - GB
    D = GA - GB
    IF (DA.LE.0.0) THEN
        FX = GA
    ELSE IF (DB.LE.0.0) THEN
        FX = GB
    ELSE
        FX = GA - DA*D/(DA + DB)
    END IF
    KEY = 14
ELSE
    FX = 0.5*(GB + GA)
    KEY = 15
END IF
END IF
ELSE IF ( (CB1.OR.CA2).AND.(.NOT.SKIP) ) THEN
    DONE = .FALSE.
    IF (FXCA2+TOL.LT.FXCB1) THEN          !no problem with conflict
        IF (CB1) FX = FXCB1
        IF (CA2) FX = FXCA2
        IF (CB1.AND.CA2) THEN              !values close
            WRITE (*,*) '(ADJ_MOD) WARNING CB1 & CA2, IX,J=',IX,J
            WRITE (*,*) 'FXCB1,FXCA2', FXCB1,FXCA2
            WRITE (*,*) 'FXCB2,FXCA1', FXCB2,FXCA1
            WRITE (*,*) 'IXB,XB,FB,FXB', IXB,XB,FB,FXB
            WRITE (*,*) 'IX, X, F, FX ', IX, X, F, FXHOLD(J,IX)
            WRITE (*,*) 'IXA,XA,FA,FXA', IXA,XA,FA,FXA
            WRITE (*,*) 'GB,GB3', GB,GB3
            WRITE (*,*) 'GBMIN,GBMAX', GBMIN,GBMAX
            WRITE (*,*) 'GA,GA3', GA,GA3
            WRITE (*,*) 'GAMIN,GAMAX', GAMIN,GAMAX
        END IF
        KEY = 16
    ELSE
        IF (.NOT.CB1) THEN
            FX = FXCB1
            KEY = 17
            IF (FINALTRY) THEN
                FX = FXCA2
                KEY = 27
            END IF
        ELSE IF (.NOT.CA2) THEN
            FX = FXCA2
            KEY = 18
            IF (FINALTRY) THEN
                FX = FXCB1
            END IF
        END IF
    END IF

```

```

        KEY = 28
      END IF
    ELSE IF (FXCA2-TOL.GT.FXCB1) THEN
      DA = GA - FXCA2
      DB = FXCB1 - GB
      D = GA - GB
      IF (DA.LE.0.0) THEN
        FX = GA
      ELSE IF (DB.LE.0.0) THEN
        FX = GB
      ELSE
        FX = GA - DA*D/(DA + DB)
      END IF
      KEY = 19
    ELSE
      FX = 0.5*(GB + GA)
    END IF
  END IF
END IF

FXN(J,IX) = FX
IF (SKIP) FXN(J,IX) = FXHOLD(J,IX)
IF (FN(IX).EQ.FMIN) FXN(J,IX) = 0.0
END DO
END DO
! ~~~~~
! Determine nodes changed.

NOCHANGE = .TRUE.
DO IX = 1,NNODES
  DO J = 1,NX
    TOL = ERRTOL*(1.0 + ABS(FXN(J,IX)) + ABS(FXHOLD(J,IX)))
    TOL = MIN( TOL,1.0 )
    IF (ABS(FXN(J,IX)-FXHOLD(J,IX)).GT.TOL) THEN
      NOCHANGE = .FALSE.
    END IF
  END DO
END DO

! Determine if done.

IF (DONE) EXIT

! If not done, but no change, go to finaltry.

IF (NOCHANGE) THEN
  WRITE (*,*) '(ADJ_MOD) WARNING: NOT DONE, BUT NO CHANGE'
  IF (FINALTRY) THEN
    EXIT
  ELSE
    NITER = 0
    FINALTRY = .TRUE.
  END IF
END IF

! If not done, but maxiter reached, go to finaltry.

```

```

      IF (NITER.GE.MAXITER) THEN
        WRITE (*,*) '(ADJ_MOD) WARNING:  MAX ITERATIONS EXCEEDED'
        IF (FINALTRY) THEN
          EXIT
        ELSE
          NITER = 0
          FINALTRY = .TRUE.
        END IF
      END IF
    END IF
  END DO
END IF

```

```

! -----
! Write changed nodes.
! -----
!
! If not done, but maxiter reached, go to finaltry.

```

```

DO IX = 1,NNODES
  NOCHANGE = .TRUE.
  FDIFF = 9999.99
  FXDIFF = 9999.99
  TOL = ERRTOL*(1.0 + ABS(FN(IX)) + ABS(F0(IX)))
  TOL = MIN( TOL,1.0 )
  IF (ABS(FN(IX)-F0(IX)).GT.TOL) THEN
    NOCHANGE = .FALSE.
    FDIFF = FN(IX) - F0(IX)
  END IF
  DO J = 1,NX
    TOL = ERRTOL*(1.0 + ABS(FX0(J,IX)) + ABS(FXN(J,IX)))
    TOL = MIN( TOL,1.0 )
    IF (ABS(FXN(J,IX)-FX0(J,IX)).GT.TOL) THEN
      NOCHANGE = .FALSE.
      FXDIFF(J) = FXN(J,IX) - FX0(J,IX)
    END IF
  END DO
  IF (.NOT.NOCHANGE) THEN
    write (*,'(a36,i6,3f12.2,40f10.2)')
+   '(adj_mod) ix,f(o/d/n),x,fx(o/d/n):',
+   ix,      f0(ix),      fdiff, fn(ix),
+   xn(:,ix),fx0(:,ix),fxdiff,fxn(:,ix)
  END IF
END DO

```

```

      WRITE (*,*) '(ADJ_MOD) END'

```

```

END

```

Subroutine IDNOW

This routine identifies the year and season of the current stage.

```

SUBROUTINE IDNOW      ( ISTAGE, IFIRST, IYFIRST, NSEAS,
+                      IYEAR, ISEASON )
IMPLICIT NONE
INTEGER               ISTAGE, IFIRST, IYFIRST, NSEAS

```

INTEGER IYEAR, ISEASON

!-----
!Identify year and season for current stage.
!-----

INTEGER I, IY, NY

!-----
I = ISTAGE
IY = 0
IF (I.LE.0) THEN
DO
I = I + NSEAS
IY = IY + 1
IF (I.GT.0) EXIT
END DO
END IF
NY = (IFIRST+I-2)/NSEAS
IYEAR = IYFIRST + NY - IY
ISEASON = IFIRST + I - 1 - NY*NSEAS

END

Subroutine SIZETEST

This routing verifies that the value of a user-supplied variable lies within permitted bounds.

SUBROUTINE SIZETEST (N, MINN, MAXN, TITLE)

IMPLICIT NONE

INTEGER N, MINN, MAXN

CHARACTER*20 TITLE

!-----
IF (N.LT.MINN) THEN
WRITE (*,*) TITLE, ', VIOLATES MINIMUM SIZE: ', N, '<', MINN
STOP
END IF

IF (N.GT.MAXN) THEN
WRITE (*,*) TITLE, ', VIOLATES MAXIMUM SIZE: ', N, '>', MAXN
STOP
END IF

END

6. MODEL SPECIFICATION SUBROUTINES

The following routines specify parameters and functions that define the structural model, stochastic model, and solution method. Copies of these files can be grouped in separate files, each file identifying a different system.

Subroutine CALLDP

This routine calls the main subroutine.

```
PROGRAM CALLDP
IMPLICIT NONE

!-----
!Gets optimal future cost function using GDP.
!-----

!-----

CALL DYNPROG

STOP '(CALLDP) DONE'
END
```

Subroutine SPECPROB

This routine specifies the size of a problem and the solution methods to be used.

```
SUBROUTINE SPECPROBt ( MAXSEAS,
+                      NU, NX, NW,
+                      NTLIN, NTNLN, NCLIN, NCNLN,
+                      RESTART, NSTAGES, NSEAS, LABELS,
+                      IFIRST, ILAST, IYFIRST, IYLAST, LPRINT,
+                      STOCHASTIC, GDP, NEWTON,
+                      DISCOUNT, TIGHT, FTOL, UTOL )
IMPLICIT NONE
INTEGER                MAXSEAS

INTEGER                NU, NX, NW,
+                      NTLIN, NTNLN, NCLIN, NCNLN,
+                      NSTAGES, NSEAS,
+                      IFIRST, ILAST, IYFIRST, IYLAST, LPRINT
DOUBLE PRECISION      DISCOUNT, TIGHT, FTOL, UTOL
LOGICAL                RESTART, STOCHASTIC, GDP, NEWTON
CHARACTER*10           LABELS(MAXSEAS)

!-----
!Specify dimensions of problem and parameters for stochastic model.
!-----
!Parameters of the stochastic model (e.g, for streamflow) are applied to
! the multivariate random normal variables in the transition function.
!-----
```

```

!               Other local variables.
! -----
!
! -----
! Specify dimensions of problem.
! -----
!
NU = 4           !number of decision variables.
NX = 4           !number of state variables.
NW = 2           !number of stochastic variables.
!
NTLIN = 4        !number of linear transistion equations.
NTNLN = 0        !number of non-linear transistion equations.
NCLIN = 0        !number of linear constraints.
NCNLN = 0        !number of non-linear constraints.
!
! -----
! Specify if a restart of prior run.
! -----
!
! If restart, save last stage cost func as START.DAT
RESTART = .FALSE.
!
! -----
! Specify number of stages.
! -----
!
NSTAGES = 3
!
! -----
! Specify number of seasons.
! -----
!
NSEAS = 1
!
! -----
! Specify season and year of first stage.
! -----
!
IFIRST = 1
IYFIRST = 1
!
! -----
! Specify season and year of final ctg.
! -----
!
ILAST = 1
IYLAST = 4
!
! -----
! Specify discount rate.
! -----
!
! Note: ensure dr is consistent with stage length
DISCOUNT = 0.00

```

```

! -----
! Specify season labels.
! -----

! -----
! Specify if stochastic problem.
! -----

STOCHASTIC = .true.

! -----
! Specify interpolation mode.
! -----

GDP = .true.
TIGHT = 0.0

! -----
! Specify interpolation mode and precision of
! objective function and controls.
! -----

! Objective precision used by solver for assessing
! convergence.
! Control precision used by routine that calls solver
! and tests for consistent solution on recursive
! calls of solver.
! Note that required precision will vary with solver
! and with characteristics of problem. Start with a
! small number and increase as required.
! For Newton-based solvers, higher precision does not
! require much time; but, other solvers may require
! significantly more time to achieve convergence.
! Note for NPSOL: if objective is > 1, ftol is a
! relative vs. absolute precision.

NEWTON = .true.
FTOL = 1.0E-12
UTOL = 1.0E-04
C FTOL = 1.0E-06
C UTOL = 1.0E-02

! -----
! Specify level of printing for output.
! -----

LPRINT = 0

END

```

Subroutine SPECU

This routine identifies the bounds and other parameters for decision variables. Bounds can change with the stage of a problem.

```
SUBROUTINE SPECU ( NU, ISTAGE, IYEAR, ISEASON,
```

```

+          BL, BU, UGUESS, USCALE )
IMPLICIT NONE
INTEGER          NU, ISTAGE, IYEAR, ISEASON

DOUBLE PRECISION BL(NU), BU(NU),
+               UGUESS(NU), USCALE(NU)

```

```

!-----
!Specify parameters that describe the decision variables of the system
! and bounds for the current stage.
!Also specify an initial guess for solution of decision variables and
! a characteristic length scale.
!-----
!The initial guess will be used as a starting point in the optimization
! routine. The characteristic length scale will be used to specify:
! (1) other near solutions if needed (i.e., for polytope algorithm)
! (2) interval for finite difference approximations
!Other constraints on decisions are expressed in the transistion
! equation and linear/nonlinear constraints.
!-----

!-----
!
!               -----
!               Verify number of decision variables.
!               -----
!
IF (NU.NE.4) THEN
  WRITE (*,*) '(SPECU) INCORRECT NU =', NU
  STOP
END IF

!               -----
!               Specify bounds.
!               -----

BL      =    0.0

!               -----
!               Provide initial guess of solution and length scale.
!               -----

UGUESS = 0.0

USCALE = 1.0

END

```

Subroutine SPECX

This routine specifies the bounds for state variables and the grid of discrete values used to span the domain of the cost-to-go function. Bounds and discretization can change with the stage of a problem.

```

SUBROUTINE SPECX ( MAXIDX, NX, ISTAGE, IYEAR, ISEASON,
+               BL, BU, NDX, XDX )

```

```

      IMPLICIT NONE
      INTEGER          MAXIDX, NX, ISTAGE, IYEAR, ISEASON

      INTEGER          NDX(NX)
      DOUBLE PRECISION BL(NX), BU(NX),
+                     XDX(MAXIDX,NX)

!-----
!Specify parameters that describe the state of the system, bounds,
! and the discretization(grid) of the state space for the current stage.
!Order of state variables must agree with transx.
!-----

!
!-----
!Verify number of state variables.
!-----

      IF (NX.NE.4) THEN
        WRITE (*,*) '(SPECCLCON) INCORRECT NX =', NX
        STOP
      END IF

!
!-----
!Specify bounds.
!-----

      BL = 0.0
      BU = 12.0

!
!-----
!Specify (initial) discretization of state variables.
!-----

!
!Note: bounds on state variables are assumed to be
!the maximum domain bounded by discretization below.

      NDX(:) = 4
      XDX(1,:) = 0.0
      XDX(2,:) = 4.0
      XDX(3,:) = 8.0
      XDX(4,:) = 12.0

      END

```

Subroutine SPECW

This routine specifies the quadrature abscissas and weights applied to stochastic variables (i.e., the grid of discrete values used to span possible outcomes). Assumes that stochastic variables are independent but, otherwise, have any discrete or continuous distribution that can be approximated discretely. Abscissas and weights can change with the stage of a problem.

```

      SUBROUTINE SPECWt ( MAXIDW, NW, ISTAGE, IYEAR, ISEASON,
+                     MODELW, WMEAN, WSTDV, WSKEW,
+                     NDW, WDW, PROBW,

```

```

+                               SWLO, SWHI, PROBMIN, GAUSQUAD )

IMPLICIT NONE
INTEGER                        MAXIDW, NW, ISTAGE, IYEAR, ISEASON

INTEGER                        MODELW(NW), NDW(NW)
DOUBLE PRECISION              WMEAN(NW), WSTDV(NW), WSKEW(NW),
+                               WDW(MAXIDW,NW), PROBW(MAXIDW,NW),
+                               SWLO(NW), SWHI(NW), PROBMIN
LOGICAL                        GAUSQUAD

!-----
!Specify model and discretization of current stochastic variables.
!Also specifies method used to calculate expected values (quadrature).
!-----
!Model of stochastic variables are specified as independent random
! variables with known probability distribution. If stochastic inputs of
! the real system are correlated, this correlation must first be
! identified and a stochastic model produced that allows identification
! of stochastic variables with correlation removed.
!Discretization and weights can be determined by Gaussian Quadrature
! (recommended) or by evenly spaced values corresponding to standard
! deviations from swlo to swhi. If evenly spaced values are used,
! weights are determined by interpolation of the multivariate normal
! distribution using the trapezoidal rule.
!-----
!Stochastic distribution for each stochastic variable should be
! specified by MODELW (for model of distribution) and by WMEAN, WSTDV,
! and WSKEW (for first three moments). Available models are:
!   1: normal (Gaussian) (2-parameter model)
!   na 2: lognormal (2-parameter model)
!   na 3: 3-parameter lognormal (WSKEW = C)
! For two-parameter models, the third parameter is ignored. Generally,
! the third parameter is used for the skew of the distribution (except
! for three-parameter lognormal distribution).
!If desired model not available, abscissas and weights used to calculate
! expected values can be specified directly.
!-----

DOUBLE PRECISION              FACTOR

!-----
!
!-----
!Verify number of stochastic variables.
!-----

IF (NW.NE.2) THEN
  WRITE (*,*) '(SPECW) INCORRECT NW =', NW
  STOP
END IF

!-----
!For each season and each stochastic variable, specify distribution.
!-----

!Currently unable to calculate Gaussian quadrature locations and weights
! internally from distribution. Must provide these in WDW and PROBW.

```

```

!           If a stochastic variable is normally distributed and
!           Gaussian Quadrature is used, 2 discrete values will
!           generally be sufficient.

```

```

MODELW(1) = 1           !streamflow 1
WMEAN(1) = 2.0
WSTDV(1) = 0.5

```

```

NDW(1) = 2
WDW(1,1) = 1.5
WDW(2,1) = 2.5
PROBW(1,1) = 0.5
PROBW(2,1) = 0.5

```

```

MODELW(2) = 1           !streamflow 2
WMEAN(2) = 4.0
WSTDV(2) = 0.75

```

```

NDW(2) = 2
WDW(1,2) = 3.25
WDW(2,2) = 4.75
PROBW(1,2) = 0.5
PROBW(2,2) = 0.5

```

```

!           -----
!           For each season, specify distribution.
!           -----

```

```

SELECT CASE (ISEASON)

```

```

CASE (13:)

```

```

WRITE (*,*) '(TRANSW) SELECTED INVALID SEASON =', ISEASON
STOP '(SPECW)'

```

```

CASE (:0)

```

```

WRITE (*,*) '(TRANSW) SELECTED INVALID SEASON =', ISEASON
STOP '(SPECW)'

```

```

END SELECT

```

```

!           -----
!           Specify if Gaussian Quadrature is to be used.
!           -----

```

```

!           If gausquad = true, then sdlo, sdhi, and probmin
!           are not used.

```

```

GAUSQUAD = .true.

```

```

!           -----
!           Specify discretization of stochastic variables.
!           -----

```

```

SWHI = 1.645

```

SWLO = -1.645

```
! -----
! Specify min probability weight used.
! This will also be used as the maximum deviation from
! 1.0 of summation over all probability weights.
! -----
```

PROBMIN = 0.0001

END

Subroutine TRANSX

This routine identifies the state transition function for the current stage.
Derivatives should also be provided to avoid calculating finite difference estimates.

```
SUBROUTINE TRANSXt ( LEVEL, NU, NX, NW, NTNLN,
+                  ISTAGE, IYEAR, ISEASON, U, X, W,
+                  S, Y, YU, YX, YW )
  IMPLICIT NONE
  INTEGER          LEVEL, NU, NX, NW,
+                  NTNLN, ISTAGE, IYEAR, ISEASON
  DOUBLE PRECISION U(NU), X(NX), W(NW)

  DOUBLE PRECISION S(NW), Y(NX),
+                  YU(NX,NU), YX(NX,NX), YW(NX,NW)
```

```
! -----
! Specifies transistion function  $y = T(u,x,w)$  and derivatives.
! -----
```

```
! State variables must be arranged so that linear transistion functions
! come before non-linear.
```

```
! On input, LEVEL identifies partial derivatives needed. Can be used to
! avoid unnecessary calculation when derivatives not needed.
```

```
! 0: none (only y)
```

```
! 1:  $dy/du$ 
```

```
! 2:  $dy/du, dy/dx, dy/dw$ 
```

```
! On output, LEVEL identifies derivatives actually calculated.
```

```
! Derivatives that are needed but not calculated here will be
! approximated by finite difference.
```

```
! -----
! Notes: Do not include state constraints to condition y.
```

```
! State constraints are incorporated by specLCon.
```

```
! Include the effect of state variables on prediction of
! stochastic inputs here; stochastic variables w are independent
! of x.
```

```
! -----
! Other local variables.
```

```
  INTEGER          J
```



```
!           Verify parameters are consistent with current model.
!           -----
```

```
IF (NU.NE.4) THEN
  WRITE (*,*) '(SPECLCON) INCORRECT NU =', NU
  STOP
END IF
IF (NX.NE.4) THEN
  WRITE (*,*) '(SPECLCON) INCORRECT NX =', NX
  STOP
END IF
IF (NW.NE.2) THEN
  WRITE (*,*) '(SPECLCON) INCORRECT NW =', NW
  STOP
END IF
IF (NTNLN.NE.0) THEN
  WRITE (*,*) '(SPECLCON) INCORRECT NTNLN =', NTNLN
  STOP
END IF
```

```
!           -----
!           Set desired parameters.
!           -----
```

```
!           -----
!           Model stochastic inputs.
!           -----
```

```
!           Variable s allows clearer identification of
!           actual stochastic inputs without auto-correlation
!           and cross-correlation removed. Also allows
!           forecast generation.
```

```
S(1) = W(1)
```

```
!           -----
!           Identify transition functions and gradients,
!            $yx(i,j) = dy(i)/dx(j)$  (also for  $yu,yw$ ).
!           Note: the first (NX-ntnlN) transition function are
!           used to create linear constraint equations.
!           -----
```

```
YU = 0.0
YX = 0.0
YW = 0.0
```

```
Y(1) = X(1) - U(1) + W(1)
YX(1,1) = 1.0
YU(1,1) = -1.0
YW(1,1) = 1.0
```

```
Y(2) = X(2) - U(2) + W(2)
YX(2,2) = 1.0
YU(2,2) = -1.0
YW(2,2) = 1.0
```

```
Y(3) = X(3) - U(3) + U(2)
```

```

YX(3,3) = 1.0
YU(3,3) = -1.0
YU(3,2) = 1.0

Y(4) = X(4) - U(4) + U(3) + U(1)
YX(4,4) = 1.0
YU(4,4) = -1.0
YU(4,3) = 1.0
YU(4,1) = 1.0

END

```

Subroutine SPECLCON

This routine identifies linear constraints other than the bounds on decision variables and state variables.

```

SUBROUTINE SPECLCON ( NU, NX, NW, NCLIN,
+                     ISTAGE, IYEAR, ISEASON,
+                     A, BL, BU )
  IMPLICIT NONE
  INTEGER             NU, NX, NW, NCLIN,
+                     ISTAGE, IYEAR, ISEASON

  DOUBLE PRECISION    A(NCLIN,NU+NX+NW),
+                     BL(NCLIN), BU(NCLIN)

!-----
!Specify linear constraints as a linear function of decision variables,
! state variables, and stochastic variables for the current stage:
!           bl <= A*[u,x,w] <= bu
!Constraints can be equality or inequality as specified by bl and bu.
!-----
!For equality constraints, bl=bu.
!Constraint coefficients in A must be in the order:
!  (1) decision variables
!  (2) state variables
!  (3) stochastic variables
!-----

!           Controls u represent management decision.  Because
!           actual realization of stochastic variables may
!           affect feasibility of those decision, it may be
!           desirable to allow adjustment of actual controls
!           applied for certain realizations of stochastic
!           variables (e.g., an insufficient release decision
!           that results in a reservoir spilling, effectively
!           changing the actual control by causing an increase
!           in the release.  These adjustments are made in the
!           transition function specified in trans.f.

!           This can cause problems with finding opt solution:
!           (1) The resulting objective will usually not be
!               convex unless a sufficient penalty is applied to
!               deviations from the management decisions.
!           (2) The resulting objective will not be smooth,
!               resulting in poorer convergence of the solver.

```

```

!                                     Therefore, this is not allowed.
!
!                                     -----
!                                     Verify number of decision and state variables.
!                                     -----
!
      IF (NU.NE.4) THEN
        WRITE (*,*) '(SPECLCON) INCORRECT NU =', NU
        STOP
      END IF
      IF (NX.NE.4) THEN
        WRITE (*,*) '(SPECLCON) INCORRECT NX =', NX
        STOP
      END IF
      IF (NW.NE.2) THEN
        WRITE (*,*) '(SPECLCON) INCORRECT NW =', NW
        STOP
      END IF
      IF (NCLIN.NE.0) THEN
        WRITE (*,*) '(SPECLCON) INCORRECT NCLIN =', NCLIN
        STOP
      END IF

!                                     -----
!                                     Set A, matrix of linear constraint coefficients.
!                                     -----
!
!                                     -----
!                                     Set bounds on linear and nonlinear constraints.
!                                     -----
!

      END

```

Subroutine COST_NOW

This routine calls identifies the current-cost as a function of decision variables U, state variables X, and stochastic variables W.

```

      SUBROUTINE COST_NOW ( LEVEL, NU, NX, NW,
+                           ISTAGE, IYEAR, ISEASON, U, X, W,
+                           C, CU, CX, PEN )

      IMPLICIT NONE
      INTEGER          LEVEL, NU, NX, NW,
+                     ISTAGE, IYEAR, ISEASON
      DOUBLE PRECISION U(NU), X(NX), W(NW)

      DOUBLE PRECISION C, CU(NU), CX(NX), PEN

!-----
!Returns cost of decisions u given initial state x and stage istage.
!-----
!On input, LEVEL identifies derivatives needed. Can be used to avoid
! unnecessary calculation when derivatives not needed.
! 0: only c

```

```

! 1: c, dc/du
! 2: c, dc/du, dc/dx
!Note that derivatives dc/dw are not needed; w is independent of u, x.
!On output, LEVEL identifies derivatives actually calculated.
! Needed derivatives that are not calculated here will be approximated
! by finite difference.
!-----

```

```

      INTEGER          K
      DOUBLE PRECISION A(NU), DEV

```

```

!-----
!
!           -----
!           Verify inputs consistent with routine.
!           -----
!

```

```

      IF (NU.NE.4) THEN
        WRITE (*,*) '(COST_NOW) NU INCONSISTENT, ', NU
        STOP
      END IF

```

```

      IF (NX.NE.4) THEN
        WRITE (*,*) '(COST_NOW) NX INCONSISTENT, ', NX
        STOP
      END IF

```

```

      IF (NW.NE.2) THEN
        WRITE (*,*) '(COST_NOW) NW INCONSISTENT, ', NW
        STOP
      END IF

```

```

!
!           -----
!           Set desired parameters.
!           -----
!

```

```

      A(1) = 1.1
      A(2) = 1.2
      A(3) = 1.0
      A(4) = 1.3

```

```

!
!           -----
!           Calculate current cost function and derivatives.
!           -----
!

```

```

!           Calculate cost and derivatives.

```

```

      C = 0.0

```

```

      DO K = 1,NU
        DEV = U(K) - 1.0
        C = C + A(K)*DEV*DEV
        CU(K) = 2.0*A(K)*DEV
      END DO

```

```

      CX = 0.0

```

```

!           -----

```

```

!           Set penalty cost for violating a constraint.
!           -----
!
!           Penalty is only applied when using polytope solver
!           (i.e., not gdp).
!           Value should be great enough to force optimal
!           controls into feasible region.

```

```

PEN = 1000.0

```

```

END

```

Subroutine FINALCTG

This routine calls identifies the final (or terminal) cost as a function of the final state X.

```

SUBROUTINE FINALCTG ( NX, X,
+                   F, FX)

IMPLICIT NONE
INTEGER           NX
DOUBLE PRECISION  X(NX)
DOUBLE PRECISION  F, FX(NX)

!-----
!Provides use specified function value f(x) and gradient fx = df/dx for
! the final-stage future-cost function.
!-----

INTEGER           J
DOUBLE PRECISION  A(NX)

!-----

!           -----
!           Verify inputs consistent with routine.
!           -----

IF (NX.NE.4) THEN
  WRITE (*,*) '(COST_NOW) NX INCONSISTENT, ', NX
  STOP
END IF

!           -----
!           Set desired parameters.
!           -----

A(1) = 5.0
A(2) = 5.0
A(3) = 5.0
A(4) = 7.0

!           -----
!           Calculate current cost function.
!           -----

```

```

F = 0.0
FX = 0.0
DO J = 1,NX
  F = F + (X(J) - A(J))**2
  FX(J) = 2.0*(X(J) - A(J))
END DO

END

```

Subroutine SPECF

This routine provides an opportunity to adjust solution values during each stage. Adjustments can be used to partially correct errors that may accumulate. This may be useful to cut off oscillations of the interpolation.

```

SUBROUTINE SPECF      ( NX, Istage, Iyear, Iseason,
+                      Fmin, Fmax, FXmin, FXmax )

IMPLICIT NONE
INTEGER              NX, Istage, Iyear, Iseason
DOUBLE PRECISION     Fmin, Fmax, FXmin(NX), FXmax(NX)

!-----
!Specify maximum and minimum function values and gradients.
!Calculated values are adjusted based on values provided here and
! output is provided whenever adjustments needed.
!-----

!-----
!
!                      -----
!                      Verify inputs consistent with routine.
!                      -----
!

IF (NX.NE.4) THEN
  WRITE (*,*) '(SPECF) NX INCONSISTENT, ', NX
  STOP
END IF

!
!                      -----
!                      Check and modify control and cost function values.
!                      -----
!

END

```

Subroutine OUTSTAGE

This routine provides an opportunity for output of data during each stage. Data is accessed through common arrays.

```

SUBROUTINE OUTSTAGE ( NU, NX, NW, NSEAS, Istage, Iyear, Iseason,
+                   XBL, XBU )

```

```

      IMPLICIT NONE
      INTEGER          NU, NX, NW, NSEAS,
+                     ISTAGE, IYEAR, ISEASON
      DOUBLE PRECISION XBL(NX), XBU(NX)

!-----
!Provides opportunity to prepare and write output at each stage.
!-----

      LOGICAL          P

!-----
!
!               -----
!               Create flag to ensure update of routine.
!               -----
!

      IF (NU.NE.4) THEN
        WRITE (*,*) '(OUTPUT) ROUTINE NOT ADAPTED TO NU=', NU
        STOP
      END IF

      IF (NX.NE.4) THEN
        WRITE (*,*) '(OUTPUT) ROUTINE NOT ADAPTED TO NX=', NX
        STOP
      END IF

      IF (NSEAS.NE.1) THEN
        WRITE (*,*) '(OUTSTAGE) ROUTINE NOT ADAPTED TO NSEAS=', NSEAS
        STOP
      END IF

!
!               -----
!               Specify stages for printout.
!               -----
!

      P = .FALSE.

      END

```

Subroutine OUTFINAL

This routine provides an opportunity for final output of data, including 2-D grids of control policy decisions or of the cost-to-go. Data is accessed through common arrays.

```

      SUBROUTINE OUTFINAL ( NX, NW, ISTAGE, IYEAR, ISEASON,
+                          XBL, XBU )

      IMPLICIT NONE
      INTEGER          NX, NW, ISTAGE, IYEAR, ISEASON
      DOUBLE PRECISION XBL(NX), XBU(NX)

!-----
!Solves grid of controls u(x) and cost F(x) for stochastic variables w.
!If level = 1, also solves grid of derivatives dF/dx.
!-----

```

```

!-----
!
!           Variables needed by gridsolv.
!
!           INTEGER          LEVEL, J1, J2, N1, N2
!           DOUBLE PRECISION XGLO(NX), XGHI(NX), W(NW)
!           LOGICAL          P
!
!           Other local variables.
!
!           EXTERNAL          GRIDSOLV
!
!-----
!
!           -----
!           Verify parameters are consistent with current model.
!           -----
!
!           IF (NX.NE.4) THEN
!             WRITE (*,*) '(SPECLCON) INCORRECT NX =', NX
!             STOP
!           END IF
!
!           IF (NW.NE.2) THEN
!             WRITE (*,*) '(SPECLCON) INCORRECT NW =', NW
!             STOP
!           END IF
!
!           -----
!           Specify interpolated grid.
!           -----
!
!           Grid spans (j1,j2) dimensions between xglo:xghi
!           at (n1,n2) points.
!           Grids evaluated for each combination of xglo:xghi
!           for j not in {j1,j2}.
!
!           P = .FALSE.
!           LEVEL = 0
!
!           Specify dimensions and discretization.
!
!           J1 = 5
!           J2 = 3
!           N1 = 11
!           N2 = 21
!
!           Specify random variables.
!           Specify bounds
!           Get grid.
!
!           XGLO = XBL
!           XGHI = XBU
!           W(1) = 600.0
!           CALL GRIDSOLV (P,LEVEL,J1,J2,N1,N2,XGLO,XGHI,W)
!
!           END

```


REFERENCES

- Andricevic, R., and P. K. Kitanidis, Optimization of the pumping schedule in aquifer remediation under uncertainty, *Water Resources Research*, 26(5), 875-885, 1990.
- Atwood, D. F., and S. M. Gorelick, Hydraulic Gradient Control for Groundwater Contaminant Removal, *Journal of Hydrology*, 76(1/2), 85-106, 1985.
- Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
- Bellman, R. E., and S. E. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, Princeton, 1962.
- Bender, M., and S. Simonovic, Time-series modeling for long-range stream-flow forecasting, *Journal of Water Resources Planning and Management*, 126(6), 857-870, 1994.
- Benders, J. F., Partitioning procedures for solving mixed variables programming problems, *Numerical Mathematics*, 4, 238-252, 1962.
- Bhaskar, N. R., and E. E. Whitlatch, Jr., Derivation of monthly reservoir release policies, *Water Resources Research*, 16(6), 987-993, 1980.
- Bhaskar, N. R., and E. E. Whitlatch, Comparison of Reservoir Linear Operation Rules Using Linear and Dynamic Programming, *Water Resources Bulletin*, 23(6), 1027-1036, 1987.
- Billings, R. B., and D. E. Agthe, Price Elasticities for Water: A Case of Increasing Block Rates, *Land Economics*, 56, 73-84, 1980.
- Birge, J. R., Models and model value in stochastic programming, *Annals of Operations Research*, 59, 1-18, 1995.
- Bogle, M. G. V., and M. J. O'Sullivan, Stochastic Optimization of a Water Supply System, *Water Resources Research*, 15(4), 778-786, 1979.
- Bras, R. L., *Hydrology*, Addison-Wesley, Reading MA, 1990.
- Bras, R. L., R. Buchanan, and K. C. Curry, Real time adaptive closed loop control of reservoirs with the High Aswan Dam as a case study, *Water Resources Research*, 19(1), 33-52, 1983.
- Bredehoeft, J. D., E. G. Reichard, and S. M. Gorelick, If it works, don't fix it: benefits from regional groundwater management, in *Groundwater Models for Resources Analysis and Management*, A. I. El-Kadi ed., pp. 101-121, CRC Press, Boca Raton, 1995.

- Brookshire, D. S., L. S. Eubanks, and C. F. Sorg, Existence values and normative economics: Implications for valuing water resources, *Water Resources Research*, 22(11), 1509-1518, 1986.
- Buras, N., Conjunctive Operation Of Dams and Aquifers, *Journal of the Hydraulics Division Proceedings of the American Society of Civil Engineers*, 89, 111-131, 1963.
- Buras, N., *Scientific Allocation Of Water Resources, Water Resources Development and Utilization - a Rational Approach*, Elsevier, New York, 1972.
- Burges, S. J., and R. Maknoon, A systematic examination of issues in conjunctive use of ground and surface waters, Charles W. Harris Hydraulics Laboratory, University of Washington, Department of Civil Engineering, 44, 1975.
- Cameron, T. A., and M. B. Wright, Determinants of Household Water Conservation Retrofit Activity: A Discrete Choice Model Using Survey Data, *Water Resources Research*, 26(2), 179-188, 1990.
- Chang, L. C., C. A. Shoemaker, and P. L. F. Liu, Optimal Time-Varying Pumping Rates for Groundwater Remediation: Application of a Constrained Optimal Control Algorithm, *Water Resources Research*, 28(12), 3157-3173, 1992.
- Chow, V. T., D. R. Maidment, and G. W. Tauxe, Computer Time and Memory Requirements For DP and DDDP In Water Resource Systems Analysis, *Water Resources Research*, 11(5), 621-628, 1975.
- Curry, G. L., J. C. Helm, and R. A. Clark, Chance-constrained model of system of reservoirs, *Journal of the Hydraulic Division, American Society of Civil Engineering*, 99(HY12), 2353, 1973.
- CUWA, *Cost of Industrial Water Shortages*, Spectrum Economics, Inc. for California Urban Water Agencies, San Francisco, CA, 1991.
- Dandy, G., Assessing the Economic Cost of Restrictions on Outdoor Water Use, *Water Resources Research*, 28(7), 1759-1766, 1992.
- Danielson, L. E., An analysis of residential demand for water using micro time-series data, *Water Resources Research*, 15(4), 763-767, 1979.
- Dantzig, G. B., and P. W. Glynn, Parallel processors for planning under uncertainty, *Annals of Operations Research*, 22, 1-21, 1990.
- Davis, *Interpolation and Approximation*, Dover Publications, Toronto, Ontario, 1975.
- DWR, San Joaquin County Ground Water Investigation, California Department of Resources, Bulletin No. 146, 1967.
- EBMUB, Updated Water Supply Management Program, East Bay Municipal Utility District, 1992.

- Ecker, J. G., and M. Kupferschid, *Introduction to Operations Research*, Krieger Publishing, Malabar, Florida, 1991.
- Esmail Beik, S., and Y. S. Yu, Optimal Operation of Multipurpose Pool of Elk City Lake, *Journal of Water Resources Planning and Management*, 110(1), 1-14, 1984.
- Fiering, M. B., and B. B. Jackson, Synthetic Streamflows, American Geophysical Union, Water Resources Monograph 1, 1971.
- Fisher, A., D. Fullerton, N. Hatch, and P. Reinelt, Alternatives for managing drought: A comparative cost analysis, *Journal of Environmental Economics and Management*, 29(3), 304-320, 1995.
- Foster, H. S., Jr., and B. R. Beattie, Urban Residential Demand for Water in the United States, *Land Economics*, 55(1), 43-58, 1979.
- Foufoula Georgiou, E., Convex Interpolation for Gradient Dynamic Programming, *Water Resources Research*, 27(1), 31-36, 1991.
- Foufoula Georgiou, E., and P. K. Kitanidis, Gradient Dynamic Programming for Stochastic Optimal Control of Multidimensional Water Resources Systems, *Water Resources Research*, 24(8), 1345-1359, 1988.
- French, M. N., W. F. Krajewski, and R. R. Cuykendal, Rainfall forecasting in space and time using a neural network, *Journal of Hydrology*, 137, 1-37, 1992.
- Gal, S., Optimal Management of a Multireservoir Water Supply System, *Water Resources Research*, 15(4), 737-749, 1979.
- Gallager, A. R., and R. W. Robinson, Influence of Metering, Pricing, Policies and Incentives on Water Use Efficiency, Australian Water Resources Council, Technical Paper No 72, 1977.
- Georgakakos, A. P., Extended Linear Quadratic Gaussian Control: Further Extensions, *Water Resources Research*, 25(2), 191-201, 1989a.
- Georgakakos, A. P., Value of streamflow forecasting in reservoir operation, *Water Resources Bulletin*, 25(4), 789-800, 1989b.
- Georgakakos, A. P., and D. A. Vlatas, Stochastic Control of Groundwater Systems, *Water Resources Research*, 27(8), 2077-2090, 1991.
- Gilbert, J. B., Preparing short- and long-term measures for mitigation of droughts, *Water Supply*, 5(1), 101-110, 1986.
- Gill, P. E., W. Murray, M. A. Saunders, and M. H. Wright, User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming, Department of Operations Research, Stanford University, Technical Report SOL 86-2, 1986.

- Glantz, M. H., Consequences and responsibilities in drought forecasting; the case of Yakima, 1977, *Water Resources Research*, 18(1), 3-13, 1982.
- Gorenstin, B. G., N. M. Campodonico, J. P. da Costa, and M. V. F. Pereira, Stochastic optimization of a hydro-thermal system including network constraints, *IEEE Transactions on Power Systems*, 7(2), 791-7, 1992.
- Hanke, S. H., A Cost-Benefit Analysis of Water Use Restrictions, *Water Supply and Management*, 4(4), 269-274, 1980.
- Headley, C. J., The Relation Of Family Income and Use Of Water For Residential and Commercial Purposes In the San Francisco-Oakland Metropolitan Area, *Land Economics*, 39(4), 441-49, 1963.
- Heidari, M., V. E. N. Te Chow, P. V. Kokotovic, and D. D. Meredith, Discrete Differential Dynamic Programming Approach to Water Resources Systems Optimization, *Water Resources Research*, 7(2), 273-282, 1971.
- Hillier, F. S., and G. J. Lieberman, *Introduction to Operations Research*, McGraw-Hill, New York, 1990.
- Howitt, R. E., Empirical analysis of water market institutions: The 1991 California water market, *Resource and Energy Economics*, 16, 357-371, 1994.
- Infanger, G., Monte Carlo (importance) sampling within a Benders decomposition algorithm for stochastic linear programs, *Annals of Operations Research*, 39, 69-95, 1991.
- Jacobs, J., G. Freeman, J. Grygier, D. Morton, G. Schultz, K. Staschus, and J. Stedinger, SOCRATES: a system for scheduling hydroelectric generation under uncertainty, *Annals of Operations Research*, 59, 99-133, 1995.
- Johnson, S. A., J. R. Stedinger, and C. A. Shoemaker, Computational improvements in dynamic programming, *Forefronts*, 4(7), 3-7, 1988.
- Johnson, S. A., J. R. Stedinger, C. A. Shoemaker, Y. Li, and J. A. Tejada-Guibert, Numerical solution of continuous-state dynamic programs using linear and spline interpolation, *Operations Research*, 41(3), 484-500, 1993.
- Johnson, S. A., J. R. Stedinger, and K. Staschus, Heuristic Operating Policies for Reservoir System Simulation, *Water Resources Research*, 27(5), 673-685, 1991.
- Jones, L., R. Willis, and W. W. G. Yeh, Optimal Control of Nonlinear Groundwater Hydraulics Using Differential Dynamic Programming, *Water Resources Research*, 23(11), 2097-2106, 1987.
- Karamouz, M., and M. H. Houck, Annual and Monthly Reservoir Operating Rules Generated By Deterministic Optimization, *Water Resources Research*, 18(5), 1337-1344, 1982.

- Karamouz, M., and M. H. Houck, Comparison of Stochastic and Deterministic Dynamic Programming for Reservoir Operating Rule Generation, *Water Resources Bulletin*, 23(1), 1-9, 1987.
- Karamouz, M., M. H. Houck, and J. W. Delleur, Optimization and Simulation of Multiple Reservoir Systems, *Journal of Water Resources Planning and Management*, 118(1), 71-81, 1992.
- Karamouz, M., and H. V. Vasiliadis, Bayesian Stochastic Optimization of Reservoir Operation Using Uncertain Forecasts, *Water Resources Research*, 28(5), 1221-1232, 1992.
- Karunanithi, N., W. J. Grenney, D. Whitley, and K. Bovee, Neural networks for river flow prediction, *Journal of Computing in Civil Engineering*, 8(2), 201-220, 1994.
- Keeping, E. S., *Introduction to Statistical Inference*, Dover Publications, New York, 1995.
- Kelman, J., J. R. Stedinger, L. A. Cooper, E. Hsu, and S. Q. Yuan, Sampling Stochastic Dynamic Programming Applied to Reservoir Operation, *Water Resources Research*, 26(3), 447-454, 1990.
- Kitanidis, P. K., Real-Time Forecasting of River Flows and Stochastic Optimal Control of Multireservoir Systems, Iowa Institute of Hydraulic Research, The University of Iowa, IIHR report no 258, ISWRR completion report no 133, 1983.
- Kitanidis, P. K., Hermite Interpolation on an n -dimensional rectangular grid, St. Anthony Falls Hydraulics Laboratory, Univ. of Minn., 1986.
- Kitanidis, P. K., A first-order approximation to stochastic optimal control of reservoirs, *Stochastic Hydrology and Hydraulics*, 1(3), 169-184, 1987.
- Kitanidis, P. K., and R. Andricevic, Accuracy of the first-order approximation to the stochastic optimal control of reservoirs, in *Dynamic Programming for Optimal Water Resources Systems Analysis*, A. O. Esogbue ed., pp. 373-385, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- Kitanidis, P. K., and E. Foufoula Georgiou, Error Analysis of Conventional Discrete and Gradient Dynamic Programming, *Water Resources Research*, 23(5), 845-858, 1987.
- Lamond, B. F., and A. Boukhtouta, Optimizing future hydro-power production using Markov decision processes, Laval University, Document of Work 95-44, 1995.
- Larson, R. E., *State Increment Dynamic Programming*, Elsevier, New York, 1968.
- Lee, S. I., and P. K. Kitanidis, Optimal Estimation and Scheduling in Aquifer Remediation with Incomplete Information, *Water Resources Research*, 27(9), 2203-2217, 1991.

- Lettenmaier, D. P., and S. J. Burges, Reliability of cyclic surface and groundwater storage systems for water supply: A preliminary assessment, University of Washington, Department of Civil Engineering, Technical Report 64, 1979.
- Loaiciga, H. A., and M. A. Marino, An Approach to Parameter Estimation and Stochastic Control in Water Resources With an Application to Reservoir Operation, *Source Water Resources Research*, 21(11), 1575-1585, 1985.
- Loucks, D. P., and P. J. Dorfman, An evaluation of some linear decision rules in chance-constrained models for reservoir planning and operation, *Water Resources Research*, 11(6), 777-782, 1975.
- Loucks, D. P., J. R. Stedinger, and H. A. Haith, *Water resource systems planning and analysis*, Prentice-Hall, Engelwood Cliffs, NJ, 1981.
- Major, D. C., Multi-Objective Water Resources Planning, American Geophysical Union, Water Resources Monograph 4, 1977.
- Marien, J. L., J. M. Damazio, and F. S. Costa, Building flood control rule curves for multipurpose multireservoir systems using controllability conditions, *Water Resources Research*, 30(4), 1135-1144, 1994.
- Marino, M. A., and S. P. Simonovic, Single Multipurpose Reservoir Design: A Modified Optimal Control Problem by Chance-Constrained Programming, *Advances in Water Resources*, 4(1), 43-48, 1981.
- Martin, W. E., and S. Kulakowski, Water Price as a Policy Variable in Managing Urban Water Use: Tucson, Arizona, *Water Resources Research*, 27(2), 157-166, 1991.
- Martin, W. E., and J. F. Thomas, Policy Relevance in Studies of Urban Residential Water Demand, *Water Resources Research*, 22(13), 1735-1741, 1986.
- Max, J., Quantizing for minimizing distortion, *IRE Transactions on Information Theory*, IT6, 7-12, 1960.
- McClurg, S., Unresolved Issues in Water Marketing, Western Water, May/June, 4-11, Water Education Foundation, 1992a.
- McClurg, S., Urban Water Costs, *Western Water*, March/April, 4-11, 1992b.
- McLaughlin, D., and H. L. Velasco, Real-time control of a system of large hydropower reservoirs, *Water Resources Research*, 26(4), 623-35, 1990.
- Mercer, L. J., and W. D. Morgan, Welfare Effects of Alternative Water Rationing Schemes: a Case Study, *Source Water Resources Bulletin*, 25(1), 203-210, 1989.
- Moncur, J. E. T., Urban Water Pricing and Drought Management, *Water Resources Research*, 23(3), 393-398, 1987.
- Moncur, J. E. T., Drought Episodes Management: The Role of Price, *Water Resources Bulletin*, 25(3), 499-505, 1989.

- Murray, D., and S. Yakowitz, Constrained differential dynamic programming and its application to multireservoir control, *Water Resources Research*, 15(5), 1017-1027, 1979.
- Pereira, M. V. F., and L. M. V. G. Pinto, Application of decomposition techniques to the mid- and short-term scheduling of hydrothermal systems, *IEEE Transactions on Power Apparatus and Systems*, PAS-102(11), 3611-3618, 1983.
- Pereira, M. V. F., and L. M. V. G. Pinto, Stochastic Optimization of a Multireservoir Hydroelectric System: A Decomposition Approach, *Water Resources Research*, 21(6), 779-792, 1985.
- Pereira, M. V. F., and L. M. V. G. Pinto, Multi-stage stochastic optimization applied to energy planning, *Mathematical Programming*, 52(2), 359-375, 1991.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN: the art of scientific computing*, Press Syndicate of the University of Cambridge, Cambridge, 1992.
- Raman, H., and V. Chandramouli, Deriving a general operating policy for reservoirs using neural network, *Journal of Water Resources Planning and Management*, 122(5), 342-347, 1996.
- Ranjithan, S., J. W. Eheart, and J. H. Garrett, Jr., Neural network-based screening for groundwater reclamation under uncertainty, *Water Resources Research*, 29(3), 563-574, 1993.
- Rashid, A., A. Aziz, and K.-F. V. Wong, Neural-network approach to the determination of aquifer parameters, *Ground Water*, 30(2), 164-166, 1992.
- Revelle, C., E. Joeres, and W. Kirby, The Linear Decision Rule In Reservoir Management and Design. I. Development of the Stochastic Models, *Water Resources Research*, 5(4), 767-777, 1969.
- Rizzo, D. M., and D. E. Dougherty, Characterization of aquifer properties using artificial neural networks: Neural kriging, *Water Resources Research*, 30(2), 483-497, 1994.
- Rogers, L. L., and F. U. Dowla, Optimization of groundwater remediation using artificial neural networks with parallel solute transport modeling, *Water Resources Research*, 30(2), 457-481, 1994.
- Rogers, P. P., and M. B. Fiering, Use of Systems Analysis in Water Management, *Water Resources Research*, 22(9), 146s-158s, 1986.
- Rosa, D. J., Water pricing to achieve efficient allocation among competing users, in *Water resources planning and management and urban water resources*, J. L.

- Anderson ed., pp. 376-380, American Society of Civil Engineers, New York, 1991.
- Rotting, T. A., and A. Gjelsvik, Stochastic dual dynamic programming for seasonal scheduling in the Norwegian power system, *IEEE Transactions on Power Systems*, 7(1), 273-9, 1992.
- Saad, M., P. Bigras, A. Turgeon, and R. Duquette, Fuzzy learning decomposition for the scheduling of hydroelectric power systems, *Water Resources Research*, 32(1), 179-186, 1996.
- Saad, M., and A. Turgeon, Application of Principal Component Analysis to Long-Term Reservoir Management, *Water Resources Research*, 24(7), 907-912, 1988.
- Saad, M., A. Turgeon, P. Bigras, and R. Durquette, Learning disaggregation technique for the operation of long-term hydroelectric power systems, *Water Resources Research*, 30(11), 3195-3202, 1994.
- Saad, M., A. Turgeon, and J. R. Stedinger, Censored-Data Correlation and Principal Component Dynamic Programming, *Water Resources Research*, 28(8), 2135-2140, 1992.
- Salas, J. D., J. W. Delleur, Y. Yevjevich, and W. L. Lane, *Applied Modeling of Hydrologic Series*, Water Resources Publications, Littleton, Colorado, 1980.
- Schlette, T. C., and D. C. Kemp, Setting Rates to Encourage Water Conservation, *Water Engineering and Management*, 138, 25-29, 1991.
- Simonovic, S., Two-Step Algorithm for Design-Stage Long-Term Control of a Multipurpose Reservoir, *Advances in Water Resources*, 2(1), 47-49, 1979.
- Snedecor, G. W., and W. G. Cochran, *Statistical Methods*, Iowa State University Press, Ames, Iowa, 1989.
- Sobel, M. J., Reservoir Management Models, *Water Resources Research*, 11(6), 767-776, 1975.
- Sobel, M. J., A multi-reservoir model with a myopic optimum, in *Dynamic Programming for Optimal Water Resources Systems Analysis*, A. O. Esogbue ed., pp. 309-315, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- Stedinger, J. R., B. F. Sule, and D. P. Loucks, Stochastic Dynamic Programming Models for Reservoir Operation Optimization, *Water Resources Research*, 20(11), 1499-1505, 1984.
- Stengel, R. F., *Optimal Control and Estimation*, Dover Publications, New York, 1994.
- Tang, Z., and P. A. Fishwick, Feedforward neural nets as models for time series forecasting, *Journal of Computing*, 5(4), 374-385, 1993.

- Tejada-Guibert, J. A., S. Johnson, A., and J. R. Stedinger, Comparison of Two Approaches for Implementing Multireservoir Operating Policies Derived Using Stochastic Dynamic Programming, *Water Resources Research*, 29(12), 3969-3980, 1993.
- Tejada-Guibert, J. A., S. A. Johnson, and J. R. Stedinger, The value of hydrologic information in stochastic dynamic programming models of a multireservoir system, *Water Resources Research*, 31(10), 2571-2579, 1995.
- Terry, L. A., T. A. Pereira, T. A. Araripe Neto, L. F. C. A. Silva, and P. R. H. Sales, Coordinating the energy generation of the Brazilian national hydrothermal electrical generating system, *Interfaces*, 16, 16-38, 1986.
- Turgeon, A., Optimal Operation of Multireservoir Power Systems with Stochastic Inflows, *Water Resources Research*, 16(2), 275-283, 1980.
- Turgeon, A., A Decomposition Method For The Long-Term Scheduling Of Reservoirs in Series, *Water Resources Research*, 17(6), 1565-1570, 1981.
- Valdes, J. B., J. Montbrun-Di Filippo, K. M. Strzepek, and P. J. Restrepo, Aggregation-Disaggregation Approach to Multireservoir Operation, *Journal of Water Resources Planning and Management*, 118(4), 423-444, 1992.
- van der Leeden, F., F. L. Troise, and D. K. Todd, *The Water Encyclopedia*, Lewis Publishers, Chelsea, Michigan, 1990.
- Wagner, B. J., and S. M. Gorelick, Optimal Groundwater Quality Management Under Parameter Uncertainty, *Water Resources Research*, 23(7), 1162-1174, 1987.
- Wallis, J. R., N. C. Matalas, and J. R. Slack, Just a moment!, *Water Resources Research*, 10(2), 211-219, 1974.
- Wasimi, S. A., and P. K. Kitanidis, Real-Time Forecasting and Daily Operation of a Multireservoir System During Floods by Linear Quadratic Gaussian Control, *Water Resources Research*, 19(6), 1511-1522, 1983.
- Weiner, D., and A. Ben Zvi, A Stochastic Dynamic Programming Model for the Operation of the Mediterranean-Dead Sea Project, *Water Resources Research*, 18(4), 729-734, 1982.
- Whiffen, G. J., and C. A. Shoemaker, Nonlinear Weighted Feedback Control of Groundwater Remediation Under Uncertainty, *Water Resources Research*, 29(9), 3277-3289, 1993.
- Williams, M., and B. Suh, Demand for Urban Water by Customer Class, *Applied Economics*, 18(12), 1275-1289, 1986.
- Willis, R., and W.-G. Yeh, *Groundwater Systems Planning and Management*, Prentice Hall, Inc., Englewood Cliffs NJ, 1987.

- Wolfram, S., *Mathematica*, Addison-Wesley, Champaign, Illinois, 1991.
- Yakowitz, S., Dynamic Programming Applications in Water Resources, *Water Resources Research*, 18(4), 673-696, 1982.
- Yeh, W. W.-G., Reservoir management and operations models: A state-of-the-art review, *Water Resources Research*, 21(12), 1797-1818, 1985.
- Young, G. K., Sr., Finding Reservoir Operating Rules, *Proceedings American Society of Civil Engineering Journal Hydraulics Division*, 93(HY6), 297-321, 1967.
- Young, R. A., Price elasticity of demand for municipal water: a case study of Tucson, Arizona, *Water Resources Research*, 9(4), 1068-1072, 1973.
- Zarnikau, J., Spot market pricing of water resources and efficient means of rationing water during scarcity (water pricing), *Resource and Energy Economics*, 16(3), 189-210, 1994.